standard	RCN-218 **DCC protocol** **DCC-A – Automatic Registration**	RailCommunity
output July 27, 2025		RailCommunity – Association the manufacturer of digital Model Railway Products eV

Contents

1 General

1.1 Purpose of the standard

This standard describes DCC-A, an automatic registration procedure for DCC. This significantly increases the user-friendliness of model railway control systems. Applying this standard relieves the user of the burden of assigning addresses and functions. The goal is, for example, to have a vehicle immediately available in the control panel with its name and all its properties after it has been put on the rails.

Application of this standard provides:

- Super-fast registration – true plug & play. Unpack, put on the track, and get started. • Direct access to the decoder/ vehicle properties • Low-effort implementation for both command stations and decoders
- Compatibility with existing decoders and central units

This standard is based on the packet structures for DCC and RailCom described in the standards [RCN-211] and [RCN-217].

To support implementation in the central unit and the decoder, header files and examples for checksum calculation are available at the RailCommunity.

This standard describes only the data exchange between decoder and central unit, which is This includes the registration and the data urgently required for registration. Further data exchange then takes place based on the assigned DCC address.

1.2 Requirements

To comply with this standard, all commands and data structures specified here must be supported. Optional components are marked separately.

The correct and prompt receipt of the messages defined in this standard is essential for the registration process. To ensure this, the following requirements must be met:

- Message security according to chapter 1.4.
- Messages according to this standard may follow one another directly. A minimum interval between two messages to the same decoder, as may be defined in other standards, is not required – except for SET_DATA_END.

When marking components that comply with this standard, the uniform notation DCC-A must be used

1.3 Identification and recognition of a DCC layout

To provide decoders with a quick way to determine whether the DCC-A registration procedure is implemented on the existing track layout, all DCC messages are marked with a signature in the form of a special pulse width of the first zero after the preamble bits, i.e., the packet start bit. Central units with active DCC-A generate the DCC signal as follows:

A zero is generated symmetrically at 100us ±2us per half-bit. In contrast, the first zero after the preamble, i.e., the start bit of a DCC packet, is also generated symmetrically at 114us ±2us per half-bit.

A decoder recognizes a DCC-A system if the entire start bit is at least 20 µs longer than the other received 0 bits.

1.4 Message security

A complete evaluation of the bit timing of the DCC coding is essential for transmission reliability. The decoder must consider both edges of the DCC signal during evaluation. Messages that show errors during evaluation must be discarded.
For reasons of harmonization with [S-9.2.1.1], an additional 8-bit CRC checksum is introduced for messages. Since this results in two check bytes – the EXOR-generated check byte according to [RCN-211] and the newly introduced CRC checksum –
The terms EXOR byte (or EXOR for short) and CRC byte (or CRC for short) are used below.

The following rules apply:

a) DCC messages:
DCC messages are always and generally protected by a final XOR.
For messages conforming to this standard whose first byte is 254, a CRC is inserted before the XOR if the message (without any checksum) contains 6 or more bytes including the address 254. Messages whose CRC checksum (if any) or XOR is incorrect shall be discarded.

b) RailCom messages:
This describes the use of the CRC for each message.

c) Calculation of the CRC:
On the sender side, the CRC is calculated using the polynomial x8 + x5 + x4 + 1 over the message, starting with the first byte of the message, initialized to 0 (or the data space number, see Section 4.3), and not inverted. On the receiver side, the CRC is calculated using the same polynomial over the entire message, including the CRC; the result must be 0. For examples, informative notes, and algorithm suggestions, see Appendix C: CRC Calculation.

1.5 Glossary, definitions

The following specifications apply within this standard:

• The bits of a byte are numbered from 0 to 7. Bit 0 is the least significant bit and is located on the far right, and bit 7 is the most significant bit and is located on the far left. • A dash is inserted between bits 4 and 3 for better readability: 7654-3210.

• Apart from the description of the structure of the DCC packets in section 3, o the zeros between the bytes of a DCC packet, o the synchronization bits o the introductory byte 1111-1110 and o the check bytes CCCC-CCCC (CRC) and/or PPPP-PPPP (XOR)

not displayed. Therefore, the DCC packets are always 2 or 3 bytes longer than the displayed command. Often , the
entire byte is displayed; bits are irrelevant in the current context.
are marked with 'x'.

• The following symbols are used to indicate the meaning of a bit:

0 Bit value 0

1 bit value 1

A address bit

a Number of bytes to be transferred

B command bit

C CRC check bits

c Change count

D data bits

G Addressing group

H Manufacturer identification according to [S-9.2.2 Appendix A]

M Addressing mode N Number
of the data space

P XOR check bits

S Session ID (session number)

U Unique ID part of the manufacturer (32 bit, product identifier + serial number)

V Configuration variable bits (addressing)

x Placeholder for a bit whose value depends on the type of packet and command
and is not examined in more detail at this point.

Z ZID = Central Identification Number

• In addition, the following terms are defined:

Unique ID: The unique identifier programmed into the module (decoder/central unit) by the manufacturer, consisting of a 12-bit manufacturer ID and a 32-bit manufacturer-specific number (e.g. product index and serial number).
If an input/display is required, this should be done in the following format, where, as an exception, each letter represents a hex character (nibble), i.e. 4 bits each:

HHHUUUUUUUu,

HHH denotes the 12-bit manufacturer identifier according to [S-9.2.2 Appendix A], where the first nibble contains the top 4 bits of the manufacturer's ID,
UUUUUUUu denotes the unique identifier (as 32-bit hex), which is represented as big endian (as is typical in DCC). **Uu** therefore denotes the least significant byte.

The output should be displayed in the form V… P …. …., where the dots are replaced by the hexadecimal values for HHH and UUUUUUUU in the order shown above. (V, P stands for Vendor, Product)

ACK: Acknowledge according to [RCN-217] in the encoding 0x0F The unique ID

DID: of a decoder.

ZID: The central unit's identifier. This is a 16-bit value calculated by hashing the manufacturer's identifier and the central unit's unique ID. It allows decoders to detect a change of central unit and then re-register accordingly. See Appendix E for an example calculation of the ZID.

The value ZID = 0 is reserved.

Session ID: A variable that identifies the current operating phase. The session ID or operating phase changes as specified in Section 6.

Backoff: If a decoder does not receive confirmation after an attempted login, it will no longer respond to a (variable) number of LOGON_ENABLE messages.

Conventional address:

Indicates the address resulting from the evaluation of CVs 1, 17, 18, 19 and 29.

DCC-A Address:

Is the address assigned during operation with DCC-A.

CRC The CRC (Cyclic Redundancy Check) is a checksum calculated according to Appendix C. For DCC packets with more than six bytes, it is transmitted in addition to the check byte formed by XOR. When transmitting data spaces via RailCom, the CRC is used to secure a block with a maximum of 31 bytes of payload data, which can span multiple RailCom gaps.

XDCC: An extension of the DCC format according to NMRA standard [S-9.2.1.1], in which The decoder is addressed via address 253 followed by a packed DCC address. Generally, the decoder is addressed via address 253 based on the DCC address, and via address 254 based on the UID. RailCom responses to commands to address 253 use only channel 2, while commands to address 254 bundle both RailCom channels 1 and 2, and 6 bytes or 8 symbols are always transmitted. In both standards, DCC packets with more than 6 bytes are protected with a CRC in addition to the EXOR byte.

2 General procedure

Decoders are controlled in parallel in a DCC system. Therefore, an addressing scheme is required.
A unique identifier (Unique ID) is used to distinguish between multiple decoders. Based on this Unique ID, the decoders are assigned a shortened (session) address in order to make the most optimal use of the limited resource 'DCC bandwidth' during operation. If possible, the previous decoder address is used for the session address. For this purpose, a registration procedure is carried out at the beginning to carry out this assignment and to transfer the decoder and

To make vehicle characteristics known for vehicle control.

Automatic registration is divided into the following main phases:

• Isolation phase:
Here, the existing decoders are identified and any access conflicts that may arise are resolved. At the end of the isolation phase, the central unit knows the DIDs of the existing decoders. • Announcement phase:

Here, the central unit and decoder exchange information about the address to be used, locomotive names, available functions, etc. • Registration: The
decoder is registered
with the central unit and can then be operationally controlled ('controlled').

To speed up the registration process, the central unit can and should attempt to register decoders already known from the last operating phase directly without a separation phase.

A detailed description of the process of starting the system and starting a decoder in different operating situations can be found in [TN-218].

2.1 Separation phase

The control center sends requests to the decoders to log on (command: LOGON_ENABLE).
These login requests contain a unique identifier consisting of the control center/system ID (ZID) and a session ID. Using this identifier, the decoders can recognize the control center after a power failure. The decoders respond to the LOGON_ENABLE command with their unique ID according to specific rules.

(RailCom message ID15 Decoder-Unique registration).

If many decoders are already known in the central unit or if local RailCom detectors are used, this phase will be short. In the case of collisions between simultaneous responses from multiple decoders, detection is not reliable; in this case, a separation is performed using dynamic decoder backoffs. This separation is performed (indicated by the coding of the LOGON_ENABLE command) separately for accessory decoders and mobile decoders, or jointly.

2.2 Announcement phase

The central unit confirms the registration and addresses the decoder via its DID (commands: SELECT / GET_DATA_*). The decoder now knows that its registration has been recognized and transmits a summary of its most important control parameters, such as the desired decoder address or other information, in the corresponding RailCom response.

2.3 Registration

The central unit assigns the decoder a vehicle address to be used in this session (LOGON_ASSIGN).
The decoder responds to this message with a revision identifier (incremental number/hash of its CVs), allowing the central unit to check whether the decoder's parameters (which may already be known) are still valid or whether further reading or calibration is required.

A temporarily assigned address is only valid for that session. To permanently assign the transmitted address to CV1 or CV17/18 (and CV29 bit 5), either the corresponding DCC command must be used (preferably the short form of the configuration variable access command) or the address must be permanently assigned. Likewise, any existing multiple unit address (from CV19) loses its validity upon registration. The multiple unit only becomes valid again when it is set again with the multiple unit address command or confirmed via a POM DCC command. The assigned address applies to all vehicle commands.

The decoder must store the central unit identification (ZID), the session ID and the assigned temporary DCC-A address in a non-volatile memory to enable a quick start after a power failure.

2.4 Accelerated reading/writing of decoder parameters

The limited bandwidth of DCC and RailCom requires efficient use to transfer data from the decoder to the control center at an acceptable speed.

For this purpose, the relevant information in the decoder is transferred to data rooms according to Section 5. A complete data space can be read with the commands SELECT / GET_DATA_* or written with SELECT / SET_DATA_*.
The groups include, among other things, the protocol capabilities, basic decoder properties such as name, functions, locomotive image and direct CV access.

Any additional data should be transferred using a general transfer procedure according to [S-9.2.1.1], where both direct reading and background reading are possible.

3 DCC commands

In general, the following format applies to all DCC commands for automatic registration:

{Synchron bits} 0 1111-1110 0 {Command byte}
{[0 parameter byte(s)]} 0 {CRC 0} PPPP-PPPP 1

Generally, all automatic registration commands begin with **1111-1110.** The following command byte specifies the type of command, and further bytes define the parameters. A DCC packet can be up to 15 bytes long. Depending on the command length, a CRC is inserted (see Chapter 1.4).

3.1 Command coding

The first byte of a command determines the type of command. Below is an overview of all commands, sorted by the value of the first byte. In the following sections, the commands are described sorted by their function.

Command byte (second byte in the packet)	Length	Selection	RailCom-Answer	Meaning
0000-0000	1	Hist.	Data Stream	GET_DATA_START: Data query first transport command
0000-0001	1	Hist.	Data Stream	GET_DATA_CONT: Data query transport command
0000-0010	10	Hist. ACK		SET_DATA: Data Write Command
0000-0011	10	Hist. ACK		SET_DATA_END: Write data last command
0000-0100 to **0000-1111**	-	Hist.	no	reserved
0001-0000 to **1011-1111**	-	-	no	reserved
1100-HHHH	-	-	no	reserved for update log
1101-HHHH	7 … 12	DID ACK		SELECT: Selects the decoder and confirms registration
1110-HHHH	8	DID	ID13, State (direct Answer)	LOGON_ASSIGN: Address assignment, operating license
1111-0000 to **1111-1011**	-	-	-	reserved

1111-11xx	4	Backoff	ID15, Registration (direct Answer)	LOGON_ENABLE: Login prompt (xx = mode)

Notes:

- The Length column indicates the total command length in bytes (excluding the leading byte 0xFE and check bytes).
- The command arrangement is chosen so that within the decoder a quick selection the necessary RailCom response can be made (grouping).
- Ausw. refers to the evaluation required in the decoder: Evaluation
  - o Hist.: o and RailCom response dependent on the previous history Evaluation and RailCom
  - DID: o response only with applicable decoder unique ID
  - Backoff: Evaluation and RailCom response per command and algorithm (only if not yet registered)

3.2 LOGON_ENABLE

This command is 4 bytes long and has the format:

1111-11GG ZZZZ-ZZZZ ZZZZ-ZZZZ SSSS-SSSS

parameter	Meaning
GG	The addressing group determines which decoders are allowed to respond:
	00 ALL: All decoders
	01 LOCO: Vehicle decoder only
	10 ACC: Accessory decoder only
	11 NOW: All decoders (without considering backoff)
ZZZZ-ZZZZ ZID, first MS byte, then LS byte; central unit identification	
SSSS-SSSS Session ID (session number)	

This is the login request, which is sent periodically by the control center. The login request should be sent at least every 300 milliseconds. More frequent transmission is recommended after system startup.

In response, the decoders send their unique ID (DID) or do not respond if the decoder is currently waiting for a backoff delay. If 'NOW' is selected, all decoders not yet registered respond, and the decoder's internal backoff is then randomly reset in the range 0...7.

3.3 SELECT

The SELECT command consists of an address part (7 bytes), followed by a subcommand byte and optionally additional parameters. It has the following format:

1101-HHHH HHHH-HHHH UUUU-UUUU UUUU-UUUU UUUU-UUUU UUUU-UUUU BBBB-BBBB (NNNN-NNNN VVVV-VVVV VVVV-VVVV vvvv-vvvv aaaa-aaaa)

parameter	Meaning
HHHH...HHHH	12-bit manufacturer identification according to [S-9.2.2 Appendix A], the lower 8 bits of the manufacturer identification are in the second byte of the command.
UUUU...UUUU 32 bit, unique identifier of the decoder (part 2 of the Unique ID)	
BBBB-BBBB subcommand byte (see 3.3.1 ff)	
NNNN-NNNN	(optional, depends on **BBBB-BBBB)** Number of the requested data room
VVVV-VVVV VVVV-VVVV vvvv-vvvv	(optional, depends on **BBBB-BBBB** and **NNNN-NNNN)** CV address when accessing the CV data space. **vvvv-vvvv** denotes the low-order byte; the first **VVVV-VVVV** corresponds to CV31, the second **VVVV-VVVV** corresponds to CV32.
aaaa-aaaa	(optional, depends on **BBBB-BBBB)** Number of CVs requested

With this command, the control unit selects a decoder. This message tells decoders that their registration has been recognized and they are not allowed to send any further registrations with ID15.

In the SELECT command, the subcommand specifies the operation to be performed by the decoder. This subcommand is coded analogously to [S-9.2.1.1], although only a subset of the operations possible there are used.

Subcommand byte meaning
1111-1111 Read ShortInfo
1111-1110 Read Block
1111-1101 reserved.
1111-1100 Write Block
1111-1011 Set decoder internal status
1111-1010 … 0000-0000 reserved

The SELECT command, except for the Read ShortInfo subcommand (see 3.3.1), is acknowledged with 8 ACKs. Older decoders may already respond to the Read Block subcommand with data, which is considered a valid response. These decoders respond to a

GET_DATA_START responds to a GET_DATA_CONT, meaning the data stream begins with the data in response to the Select command, and GET_DATA_START does not reset the data space. This (deprecated) behavior must be supported.

3.3.1 Read ShortInfo subcommand

The subcommand for reading the registration summary is defined as follows:

1111-1111

After the SELECT with this read command, the decoder responds in the following cutout with the requested data ShortInfo (see Section 5 Data Spaces). There is no subsequent GET_DATA_* phase.

3.3.2 ReadBlock subcommand

The block reading subcommand is defined as follows:

1111-1110 NNNN-NNNN

parameter	Meaning
NNNN-NNNN	The number of the data room being queried.

For the definition of data spaces, see Section 5 Data Spaces.

After a SELECT with a read command ReadBlock the decoder responds in the following Cutout with 8 ACKs (in channel 1 and channel 2). The response begins with the next GET_DATA_START command. All subsequent GET_DATA_CONT commands must be used without gaps.

Data spaces generally have a variable length. For CV accesses, the command is supplemented with the parameters **VVVV-VVVV VVVV-VVVV vvvv-vvvv aaaa-aaaa .** These parameters specify the 24-bit address at which reading begins and the number of CVs to be read. If the parameters **VVVV-VVVV VVVV-VVVV vvvv-vvvv aaaa-aaaa** are sent even though they are not required, they are ignored.

In the response to subsequent GET_DATA_START/GET_DATA_CONT commands, the decoder sends the data from this data space. It automatically increments the access within the data space. The command sequence SELECT with subcommand ReadBlock, GET_DATA_START, and the required number of GET_DATA_CONT commands must always be observed. This sequence must not be interrupted by other protocols.

To summarize the sequence of commands and responses:

command	Answer	note
SELECT with ReadBlock subcommand	8 times	Data is permitted and will be used as described in 3.3.
GET_DATA_START Data from index 0 in Data room		Data transfer begins here
GET_DATA_CONT further data		Zero or multiple GET_DATA_CONT can be sent.

3.3.3 WriteBlock subcommand

The block write subcommand is defined as follows:

1111-1100 NNNN-NNNN SSSS-SSSS ssss-ssss

parameter	Meaning
NNNN-NNNN The	number of the data space to be written.
SSSS-SSSS MSB	of size without CRC
ssss-ssss LSB of	size without CRC

For the definition of data spaces, see Section 5 Data Spaces.

After a SELECT with write command WriteBlock, the decoder responds in the following cutout with 8 times ACK (in channel 1 and channel 2).

The following SET_DATA commands transfer the data to the decoder. The decoder also responds with eight ACKs. Based on the size specified in the SELECT command, the decoder knows how many SET_DATA commands (and, in the case of the last SET_DATA command, what length) to expect.

To ensure reception, a CRC is calculated for the entire process.
The complete SELECT command (without the DCC protection) as well as all contents (payload) of the SET_DATA commands are used for the CRC calculation and the obtained CRC is appended to the data.

This protection of the entire block is in addition to and independent of the protection of the individual commands defined in Chapter 1.4. The checksum procedures described there also apply to SET_DATA. See the example in Appendix F.

When the decoder has received the amount of data announced in the WriteBlock using one or more SET_DATA and has checked the CRC, it stores this data in its permanent memory (e.g. flash).

If another decoder was addressed with WriteBlock or the SET_DATA_END command was received before the announced data volume was reached, the write operation must be aborted and the memory contents remain unchanged.

Finally, the central unit sends SET_DATA_END. SET_DATA_END must not follow directly after the last SET_DATA command; that is, at least one other DCC packet must be sent in between, as defined in Chapter 3.5.

The decoder can respond to SET_DATA_END as follows, where the RailCom data is filled with 0x00:

- 0x00 = not yet finished. The control unit should wait for the end of the write process by sending further SET_DATA_END commands. After a maximum of 200 ms (from the last SET_DATA), the write should be considered a failure. The SET_DATA_END commands do not need to be sent continuously.

- 0x01 = the data was written successfully
- 0x02 = the data is incomplete, followed by the number of bytes received as 16 Bit Integer.
- 0x03 = faulty CRC, followed by the received and calculated CRC

Any other or missing answer will be considered “not finished yet.”

The command sequence SELECT with subcommand WriteBlock, the required number of SET_DATA and SET_DATA_END must always be observed.

If a data space is unknown, write-protected, or fundamentally not writable, the decoder responds with 8 NACKs.

3.3.4 Subcommand Set decoder internal status

The subcommand Set Decoder Internal Status is defined as follows:

1111-1011 BBBB-BBBB

BBBB-BBBB = **1111-1111** means deleting the change flags.

Further values for **BBBB-BBBB** are not yet defined and reserved.

The decoder responds in the immediately following cutout with ID13, DecoderState (as with the subcommand ASSIGN)

3.4 GET_DATA_*

The GET_DATA_* command is 1 byte long and has the format: **0000-000x**

There are two sub-variants of this command:

GET_DATA_START: **0000-0000**

This is the first data transport command after a decoder selection with SELECT. Further GET_DATA_CONT commands may follow.

GET_DATA_CONT: **0000-0001**

This is a data transport command. Additional GET_DATA_CONT commands may follow.

These commands are used to retrieve data if the decoder was previously selected using SELECT with the subcommand ReadBlock. The decoder returns the data of the retrieved data space as described in chapter 4.3. This command enables fast reading of blocks from data spaces. (Note: The command duration is 6.4 ms, so
a block of 31 bytes can be transmitted in approximately 32 ms). If the decoder does not offer data spaces, the implementation of GET_DATA can be omitted.

The following rules apply to the decoder:

1. The cutout after GET_DATA_START/GET_DATA_CONT may only be used
   be used if the decoder has been continuously DCC since the selection with the SELECT command
   with subcommand ReadBlock without interruption by other track formats
   received and evaluated without errors, including the command interpretation.
2. When the end of the data to be transmitted is reached, the decoder fills the data for
   This cutout is recorded with 0x00. Further GET_DATA_CONT commands are answered with 6 0x00s.

3. If a data space is empty, a HEADER with the length 0 and thus =
   0x00 and the CRC depending on the requested data room number are transmitted.
4. If the decoder does not know a data space, it responds with 8 NACKs.

3.5 SET_DATA_*

These commands are used to transport the data from the central unit to the decoder.

SET_DATA: **0000-0010**

This command has a variable length, is a maximum of 12 bytes long and has the format:

0000-0010 {DDDD-DDDD}

parameter	Meaning
DDDD-DDDD Up to 11 bytes of data	

A maximum of 11 data bytes are transmitted. Together with the fixed start byte = 254 for all DCC-A packets, the CRC for DCC packets with more than 6 bytes, and the DCC check byte, this results in a gross length of 15 bytes, as specified in Chapter 3.

SET_DATA_END: **0000-0011**

This command is 1 byte long and has the format:

0000-0011

This command is used for the final check of the data transfer, as described in Section 3.3.3. At least one other command (e.g., IDLE) must be inserted between the last SET_DATA and SET_DATA_END.

3.6 LOGON_ASSIGN

The LOGON_ASSIGN command is 8 bytes long and has the format:

1110-HHHH HHHH-HHHH UUUU-UUUU UUUU-UUUU UUUU-UUUU uuuu-uuuu BBAA-AAAA AAAA-AAAA

parameter	Meaning
HHHH...HHHH	12-bit manufacturer identification according to [S-9.2.2 Appendix A], the lower 8 bits of the manufacturer identification are in the second byte of the message.
UUUU...UUUU 32 bit, unique identifier of the decoder (part 2 of the Unique ID)	
BB	11 = temporary assignment of the address 10 = permanent assignment of the address The other two possible values are reserved and the message should be ignored.
AAAA...AAAA Assigned decoder address, address assignment see Appendix D	

With this command, the central unit registers the decoder. The decoder is then addressed at the transmitted address **AA-AAAA AAAA-AAAA .** The decoder may only accept the registration if it has already recorded the ZID (central unit identifier) and the session ID. If the registration is not accepted, the decoder responds with eight NACKs. The central unit should use the desired address from the ShortInfo data space, provided there is no address collision.

The decoder responds in the cutout immediately following the LOGON_ASSIGN with a message containing a summary / change index (hash) of its CVs.
This allows the control unit to check whether the decoder's parameters (which may already be known) are still valid or whether further reading actions or recalibration are required.

A temporary assignment of the address means that the conventional address in CV1 or CV17/18 (and CV29 bit 5) will not be changed. The assigned temporary address is valid as long as the decoder stores its driving information, i.e. speed, direction and functions, to bridge a power drop. If its driving information is no longer valid, it must wait for a new LOGON_ENABLE and subsequent LOGON_ASSIGN. Without driving information, however, the vehicle is stationary anyway and assignment is quick if the decoder is already known to the command station. Along with the temporary address, the decoder must also store the command station ID and session ID in order to always recognize a change of command station or a change in the session ID. The decoder may then have to log in again.

A permanent assignment means that the address is additionally stored in the conventional address in CV1 or CV17/18 (and CV29 Bit5) and the desired address in the data space ShortInfo.

Control panels should support both types of assignment and be configurable accordingly. If a permanent assignment is made, the user should receive a corresponding warning.

Assigning a short address with the value 0 means: Registration has been recognized, but the central unit does not currently want to address the decoder ('Parking').

With the automatic registration, any existing multiple traction address
is no longer observed. Only when the central unit reassigns the multiple-unit address with a multiple-unit command or POM DCC command will the multiple-unit operation be reactivated.

4 RailCom news

Generally, in responses to DCC-A messages, the two RailCom channels 1 and 2 are bundled, and 6 bytes or 8 symbols are always transmitted. However, the time distribution remains unchanged (as specified in RCN217). Bundling results in messages that always consist of 8 bytes, leaving 48 bits after the 6-8 coding.

These 48 bits are divided as follows (Big Endian):

47 – 44 43 – 40 39 – 36 35 – 32 31 – 0
ID EXT1 EXT2 EXT3 DATA

(Note: EXT1 and EXT2 are in channel 1, EXT3 leads channel 2.)

The use of the RailCom cutout plays a special role in data space transmission via GET_DATA_*. Here, the RailCom data is simply viewed as a byte sequence, with ID and EXT1 forming the first byte, EXT2 and EXT3 the second byte, DATA[31-24] the third byte, and so on.

Data is always protected with CRC, the only exception being ID15, Decoder-Unique.

4.1 ID15 - Decoder-Unique (Registration)

This message is sent in response to the DCC command LOGON_ENABLE.

ID15 - Decoder-Unique		
ID	**1111**	ID15, Decoder Unique Identifier
EXT1 **HHHH**		4-bit manufacturer identification (according to [S-9.2.2 Appendix A]), here the top 4 bits of the 12-bit manufacturer identification
EXT2, EXT3	**HHHH HHHH**	8-bit manufacturer identification (according to [S-9.2.2 Appendix A]), here the lowest 8 bits of the 12-bit manufacturer identification
DATA **UUUU...UUUU**		32-bit unique identifier of the decoder (part 2 of the Unique ID)

The 12-bit manufacturer identifier is EXT1*256+EXT2*16+EXT3. This message is not CRC-protected.

4.2 ID13 - Decoder State

This message is sent in response to the DCC command LOGON_ASSIGN and contains information about whether and which decoder configuration has been changed. It also contains additional information bits about the decoder's (protocol) capabilities. This message also confirms the successful assignment of the address.

ID13 - Decoder State		
ID	**1101**	ID13, decoder state
EXT1, EXT2 **FFFF-FFFF**		Change flags indicate modified decoder parameters. EXT1 contains the MSBs.
EXT3, DATA[31..24]	**cccc- cccc-cccc**	Changecount, 12 bits. EXT3 contains the MSBs.
DATA[23..8]	**DDDD-DDDD DDDD-DDDD**	Information about the decoder's protocol capabilities, mirror of bytes 2 and 3 of data space 0. See also Chapter 5.2. Extended Capabilities.
CRC	**CCCC-CCCC**	

The change flags provide the controlling system with information about whether relevant CVs have changed. This allows the system to quickly determine whether existing values for driving behavior and operation have remained unchanged upon subsequent login.

It is permissible to change all change flags when making a change to force a complete re-registration.

Change flags	
Bit meaning	
F0	(LSB) The last reset of the change flags was done on a control unit with a different ID.
F1	The decoder firmware has been changed.
F2	The decoder's driving behavior has changed. For example, motor parameters, acceleration, braking, speed. for accessory decoders: switching times have changed
F3	The function assignment of the decoder has changed.
F4	GUI data (such as locomotive name, locomotive image) have been changed.
F5	Multiple traction (CV19) was activated.
F6	reserved: Must be sent as 0 and will be ignored by the receiver.
F7	(MSB) The conventional address (CV 1, CV 29 Bit5 or 17 and 18) and/or data space 2 (ShortGUI, including the name of the decoder) has been changed.

The decoder state is managed by the decoder. This means that every time a CV or firmware change occurs, the corresponding change flags must be set and the counter incremented. After programming, a central unit should read the counter via LOGON_ASSIGN and save it to the unique ID to detect whether a change was made by another central unit. A change count value of 0xFFF indicates a newly programmed decoder (factory default).

The subcommand Set decoder internal status with **NNNN-NNNN** = **1111-1111** deletes all change flags. The decoder remembers the ZID of the central unit, which

has issued a delete command. If the control panel has not yet been recognized, a delete command will not be accepted.

The protocol capabilities of the decoder in DATA[23..8] are defined in the following table.

DATA[23..8]	
Bit	**meaning**
8	Decoder supports Dynamic Channel 1 usage ([RCN-217] Section 5.2.1)
9	Decoder supports Info1 (ID 3) in channel 1 ([RCN-217] section 5.2.2)
10	Decoder supports sending location information ([RCN-217] Section 5.3.1)
11	Decoder supports ID 7 Dyn DV 0 & 1 speed ([RCN-217] section 5.5)
12	decoders support ID 7 Dyn DV 7 reception statistics (ditto)
13	Decoder supports ID 7 Dyn DV 21 warning and alarm messages (ditto)
14	decoder supports ID 7 Dyn DV 26 temperature (ditto)
15	decoder supports ID 7 Dyn DV 27 direction state byte (ditto)
16	decoder supports the automatic transmission of CVs via ID 12
17	Binary State Control Command Short Form ([RCN-212] Section 2.2.5)
18	Binary State Control Command Long Form ([RCN-212] Section 2.2.6)
19	Decoder supports the package for speed, direction and functions (so-called SDF command, [RCN-212] section 2.3.7)
20	Decoder supports the programming commands for CV pairs 17 and 18 as well as 31 and 32 ([RCN-214] Section 3.2)
21	reserved: Must be sent as 0 and will be ignored by the receiver.
22	Decoder supports the special mode command ([RCN-212] section 2.2.3)
23	Decoder supports multiple commands in one package according to [S-9.2.1.1]

4.3 Data room transfer using GET_DATA_*

When transmitting data using GET_DATA_*, a transmission from the decoder is distributed over one or more cutouts, whereby the following (overall) structure of a block always applies:

HEADER [DATA] CRC

For larger data spaces, several such blocks can follow one another.

If the data size is 0, a HEADER with the length 0 and thus =
0x00 and the CRC dependent on the requested data space number are transmitted. The rext of the RailCom message is padded with 0x00, as with other responses to GET_DATA_*.

The HEADER is one byte and contains information about the following data.

HEADER		
Bit 7 **H**		0: Standard format with variable total length, using one or more cutouts. The following bits in the header definition are valid. 1: Data according to special format A with a total length of 6 bytes, where only one cutout is used.
Bit 6 **S**		Response Spontaneous (*) 0: Response message to a request 1: Message that is transmitted without request
Bit 5 **C**		0: Starting block 1: Continuation block
bit 4…0	**SIZE**	Number of useful bytes in this block. The total number of bytes is SIZE+2.

(*): Queries according to this standard always result in a response message.

In responses to data space read commands, the number of the requested data space is used as the first byte (start value) for the CRC calculation. The CRC is calculated from the start value, using the HEADER and all DATA, and is appended to the message.

If the data space is larger than 31 bytes, bit 5 in the HEADER must be set starting with the second block. The starting value of the CRC remains the data space number. If the last block contains 31 bytes of payload data, an empty continuation block with HEADER = 0x20 must be transmitted. Except for the last block, all blocks must contain 31 bytes of payload data, meaning the data is packed as best as possible. This allows the receiver to recognize that a block with a SIZE < 31 is the last block.

Two examples for the transmission of data spaces with 8 and 42 bytes.

- a, b, c, … the first bytes of a transmitted block
- … x, y, z in the second example are the last bytes of a transmitted block
- H the header
- C the CRC checksum
- 0 Byte with the value 0x00

Example with 8 bytes in a block:
"Habcde" "fghC00"

Example with 42 bytes with 31 bytes in the first and 11 bytes in the second block:
"Habcde" "fghijk" ... "stuvwx" "yzCHab" "cdefgh" "ijkC00"

The data in the last RailCom message is padded with 0x00. If further GET_DATA_CONT commands follow, the decoder responds with 6 0x00s.

Special format A:

In order to be able to act particularly quickly and efficiently when reading the Decoder ShortInfo data space, this data space is always encoded with a fixed total length of 6 bytes. In this case, the MSB of the first byte is 1. The starting value of the CRC is 0.

5 data rooms

Historically, the information in the decoder was encoded in CVs, which were scattered across a certain area and not implemented uniformly everywhere. Automatic registration requires fast and efficient transmission of this information.
These are therefore combined into data rooms, allowing the relevant information to be transported with just a few transfers.

There are 256 possible data spaces. Of these, spaces 0...7 are defined in this specification; all other data spaces are reserved. However, areas are specified for certain special functions, such as manufacturer-specific data spaces.

All data spaces are also accessible via the CV area, i.e., in programming mode, via main track programming (POM), and XPOM. For this purpose, they are mirrored in the pages addressed with CV31 = 2 (see [RCN-225]).

Whether a data space is available is indicated by the decoder in a bit field, which can itself be read as a data space. The availability of the most important data spaces (Extended Capabilities, ShortGUI, and general CV access) is already announced in the ShortInfo.

Data room	
number (dec)	**Contents**
--	ShortInfo (fixed size = 6 bytes), contains information about the protocol capabilities of the decoder and the previous DCC address of the decoder.
0	Extended Capabilities (variable size, up to 31 bytes), contains information about additional protocol capabilities of the decoder
1	SpaceInfo (variable size, up to 32 bytes), contains information about available data spaces.
2	ShortGUI (variable size, up to 28 bytes), contains information about the name, image index and functionality.
3	CV block, maximum data space size is 224, from this data space a block of maximum size 31 is read. (Variable size)
4	Icon mapping with variable size, coding according to [TN-218]
5	Long name and description with variable size
6	Product information from the decoder manufacturer as four strings with variable Size
7	Vehicle-specific data space with variable size with manufacturer name, Product number and URL for a vehicle image

At least data spaces 0 to 3 should be implemented.

Variable-length data spaces are transmitted in one or more blocks with a maximum size of 31 bytes (see Chapter 4.3). Each of these blocks is generally composed as follows:
together:

HEADER [DATA] CRC

The HEADER is defined in Section 4.3. The following description of the individual data spaces (except for the ShortInfo data space) describes only the data area.

5.1 Data Room ShortInfo

This data space is 6 bytes in size, uses the special format A and contains the essential Information on registering the decoder:

Byte	Bit	Data included
0	7	1: indicates the special format
0	6	Reserved; to be pre-assigned with 0
0	5 – 0	Vehicle address (desired address), bits 13...8 (higher-order bits) (See Appendix D: Addresses)
1	7 – 0	Vehicle address (desired address), bits 7...0 (low-order byte) (See Appendix D: Addresses) Note: by using a non-mappable desired address with the higher-order bits = 0x3F, the decoder indicates that it is currently in FW update mode and only FW update is possible.
2	7 – 0	For vehicle decoders: Highest function number used including function assignment. For standard accessory decoders: Highest switch pair number (e.g. a decoder with 4 switch pairs reports 3). For extended accessory decoders: Highest occurring term. (e.g., a decoder with the terms "stop," "go," and "slow travel" reports "2").
3	7	Decoder supports FW update via DCC-A
3	6	Decoder supports XPOM
3	5	Decoder supports SELECT + GET_DATA_* (=ReadBlock)
3	4	Decoder supports ReadBlock via XDCC ([S-9.2.1.1])
3	3	Decoder supports ReadBackground via XDCC
3	2	Decoder supports SELECT with subcommand WriteBlock
3	1	Decoder supports WriteBlock via XDCC
3	0	Reserved; to be pre-assigned with 0
4	**7**	Decoder supports Dataroom 7: Manufacturer-specific information
4	6	Decoder supports Dataroom 6: Product name

4	5	Decoder supports data room 5: Long name
4	4	Decoder supports data room 4: Assignment of function icons
4	3	Decoder supports data space 3: CV (read/write via DCC-A and/or XDCC, depending on bits 3-3, 3-4 and 3-5.)
4	2	Decoder supports Dataroom 2: ShortGUI
4	1	Decoder supports data space 1: SpaceInfo
4	0	Decoder supports Data Space 0: Extended Capabilities
5	7 – 0	CRC over bytes 0, 1, 2, 3 and 4

5.2 Dataroom 0 Extended Capabilities

This data room has a variable size and contains information about the (further)
Protocol capabilities of the decoder. The first four bytes of this data space are defined by this standard; additional bytes are reserved for future extensions. Undefined/reserved bits must be preset to 0.

To enable a low-effort implementation, these first four bytes are already included (as a copy) in the ShortInfo or Decoder State (ID13) upon registration. Byte 0 is copied to data byte 3 of the ShortInfo, byte 1 to data byte 4 of the
ShortInfo, byte 2 is mirrored to DATA[23..16] and byte 3 to DATA[15..8] of DecoderState (Chapter 4.2).

byte	Meaning
0	Byte 3 of data space ShortInfo (section 5.1), decoder has capabilities like FW update via DCC-A, XPOM, ReadBlock, XDCC (2x), Auto-CV via ID 12
1	Byte 4 of data space ShortInfo (section 5.1), indicates which of the data spaces 0 to 7 are supported
2	DATA[23..16] of decoder state, DCC capabilities such as number of functions, binary state control command, SDF and special mode command
3	DATA[15..8] of decoder state, RailCom capabilities such as dynamic channel 1 usage, ID 3 in channel 1, sending location information and certain DVs
following reserved	

5.3 Data Room 1 SpaceInfo

This data space has a variable size and contains information about available data spaces as a bit field. A 1 indicates whether the respective data space is available. Bytes after the last available data space do not need to be transferred.

byte	Meaning
0 – …	Data space availability bit field. Bit 0 in byte 0 indicates the availability of data space 0, bit 1 in byte 0 indicates the availability of data space 1, bit 0 in byte 1 indicates the availability …, of data space 8, etc.

Note: Byte 0 is mirrored in byte 4 of the ShortInfo registration information. (Section 5.1)

5.4 Data Room 2 ShortGUI

This data space is variable in size and contains information in a compressed format for simple control devices. The decoder should support all available functions up to a maximum of F68. The transferred data includes:

byte	Meaning
0 – 7	Custom decoder name, utf8 encoded. If fewer than 8 bytes are used, the remainder must be padded with 0.
8	16-bit vehicle image number (MSB) according to [TN-218]
9	16-bit vehicle image number (LSB) according to [TN-218]
10	Bits 0…3: simplified principle symbol
10	Bit 4 Data incomplete (e.g. because data from a SUSI module is still missing)
10	Bit 5: reserved: Must be sent as 0 and will be ignored by the receiver.
10	Bits 6…7: Function information F0
11 – ..	Function information F1 ... Fx: two bits per function, these provide information whether the function can be used and whether it is used in a switching or touch-sensitive manner.

If a specific vehicle is to receive a special image that is not included in the general list of image numbers in [TN-218], this image must be stored in the control center with a file name containing the unique ID. The file name begins with an 'x', followed by the unique ID in hexadecimal notation, optionally a name separated from the unique ID by an underscore '_', and an extension appropriate for the image format.

The index for the principle symbol is defined as follows:

index	principle symbol to be used (mobile decoder)	principle symbol to be used (stationary decoders)
0000 steam locomotive		Switches/switching decoders
0001 diesel locomotive		light signal
0010 Electric locomotive		Form signal
0011 diesel railcar		barriers
0100 electric multiple unit (S-Bahn, ICE)		turntable
0101 Control car, rear of train		(System) lighting
0110 Passenger cars		Traffic light
0111 Freight and freight train escort cars		
1000	Service vehicles (crane, snow blower, etc.)	
1001 Other vehicle without drive		
1010 Car: Passenger car		
1011 Car: Truck		
1100 Car: Bus		
1101 tram		
1110 Other vehicle with drive		
1111 Other		Other

The function information is coded as follows:

Bit 1, Bit 0	Meaning
00	Function is not available
01	Function is available and can be used as a switch (e.g. light)
10	Function is available and can be used by touch
11	The function is available and can be used both as a switch and as a key, e.g. because it is used differently in the decoder and a SUSI module.

5.5 Data Room 3 CV-Read

This data space has a variable size and contains the requested CV values.

Structure: [requested CV data].

Note: As with the other data spaces, the data is embedded in header and CRC.

5.6 Data Room 4 Icon Assignment

This data space has a variable size and contains information about a static assignment of function numbers to function icons. Dynamic function assignments cannot be represented here and require a different, yet undefined data structure. Therefore, in the case of dynamic functions, data space 4 is only a fallback level. Icon numbers 224 to 255 are reserved for user-defined text messages.

maximum Number of bytes	**Meaning**
1	Function number
2	Number of the icon to be assigned The table of icon numbers is in [TN-218]
1	Function number
2	Number of the icon to be assigned
	…

The number of bytes per icon is determined from the data. The exact encoding rule is defined in [TN-218].

5.7 Data Room 5 Long Name

This data space has a variable size and contains the long name and description assigned by the user. The information is sent as UTF-8 encoded strings, separated by a 0-byte. Empty strings (length = 0) are permitted, but the terminating 0-byte must be present in each string.

maximum Number of bytes	**Meaning**
63	long name
63	User Description

5.8 Data Room 6 Product Information

This data space has a variable size and contains the name of the decoder manufacturer, the product name, the hardware version and the software version. The information is

Sent as UTF-8 encoded strings, separated by a 0 byte. Empty strings (length = 0) are allowed, but the terminating 0 byte must be present in each string.

This data room is read-only.

maximum Number of bytes	Meaning
41	Manufacturer name of the decoder
41	Product name of the decoder
21	Hardware version of the decoder
21	Software version of the decoder

5.9 Data room 7 Vehicle-specific data room

This data space is variable in size and contains information about the vehicle the decoder is located in. No description is provided here, only the manufacturer's name (abbreviated if necessary) and the product number for vehicle identification by a control center that maintains a database for this manufacturer.

maximum Number of bytes	Meaning
16	Manufacturer name of the vehicle
16	Alphanumeric product number of the vehicle
92	URL for a vehicle image

All fields must be specified in UTF8 encoding. PNG or JPG are acceptable image formats.
The image should be presented with the direction of travel to the right. The top of the rail should be at 95%, and a (possibly fictitious) overhead line should be at 10% of the image's height, measured from the top.

5.10 Additional data rooms

The following additional data spaces are planned, but are not yet precisely defined in this version of the standard.

Display text	Decoder user message
ConsistPartner	Possible partner decoders for multiple traction formation

6 Implementation at headquarters

This chapter describes the behavior of central units during the registration process and how they handle special cases.

A central unit may not switch DCC-A on or off during operation. Each change of operating mode must be accompanied by a shutdown of the track voltage for at least 3 seconds.

A re-registration of all decoders can be triggered by a sudden change of the session ID by more than 1 or an interruption of the operating voltage for at least 3 seconds and an increase of the session ID by 1 during this interruption.

The session ID is increased by 1 by the central unit if

- a vehicle is deleted from the database or
- an address that has already been assigned is assigned to another decoder.

6.1 Registration

- At system startup, the control unit sends a LOGON_ENABLE(NOW) to register all decoders. Depending on the type of feedback system (local detectors/global detectors), the response will be a series of ID15 messages or detected collisions.
- The decoders announced with an ID15 message are read (SELECT & READ ShortInfo); they are now known and no longer respond to LOGON_ENABLE. • If collisions are detected, a separation is initiated: the central unit sends a

sequence of LOGON_ENABLE(ALL). Depending on the current backoff value, the decoders will separate and successfully send an ID15 message. • If no new decoder IDs are received and no collisions are detected,

are no longer recognized, the control center can switch back to LOGON_ENABLE(NOW) to enable rapid registration of newly added vehicles. Decoders known to the control center from a previous

operating phase can be read out or registered directly after the first LOGON_ENABLE for testing purposes ; this generally shortens the registration phase.

- To put vehicles into operation that are placed on the track in a section without a RailCom feedback module, a LOGON_ASSIGN command can be used even without feedback from the decoder. To do this, the user must select the vehicle from the list of vehicles known to the control center and initiate the address reassignment. The user should be able to select a desired address.

- If a changed conventional address and/or data space 2 (ShortGUI, including the name of the decoder) is detected during registration due to a set bit 7 in the change flags, a new registration may have to be carried out taking the desired address into account.

6.2 Reading decoder parameters

Reading decoder parameters depends on which reading method the decoder supports.

• The recommended procedure:

After registration, the address is assigned as soon as possible, and further information is retrieved via XDCC (see [S-9.2.1.1]). Data from the decoder currently being controlled can be read using BLOCK_READ, while other decoders can be read 'invisibly' using BACKGROUND_READ.

• Fallback 1:

Until XDCC is generally established or if identical DCC addresses are assigned (locomotive and control car), the 'ShortGUI' data space and CV reading are possible via DCC-A addressing. • Fallback 2:

If a decoder does not allow reading via XDCC or DCC-A, the (very slow) last resort is normal CV reading via POM or XPOM.
The CV pages accessible via CV 31 = 2 are reserved for this purpose. Data spaces 0 to 7 are accessed via CV 32 = 0 to CV 32 = 7 (see [RCN-225]).

6.3 Checking the DCC-A suitability of a model layout

There are faulty decoders in circulation that interfere with RCN-218 messages by continuously transmitting illegal RailCom data. Therefore, a central unit should check for such decoders before activating DCC-A. To do this, it sends the DCC command SELECT + ReadBlock with the decoder UID = 0xFFF FFFF FFFF, followed by GET_DATA_START. This should not result in a RailCom response.

If a RailCom response (possibly partial or with coding errors) is received for either of these commands, DCC-A cannot be enabled. Alternatively, the user can either disable RailCom for a decoder when it reports its address, or use a section of the layout with a local detector that only contains the new vehicle for registration.

7 Behavior of decoders

This chapter describes the behavior of decoders during registration and how they handle special cases. If automatic registration is not enabled in CV# 28 bit 7, the decoder ignores all commands to address 254 and behaves as if it were on a layout without automatic registration.

7.1 Restart

Restarting the decoder can either be a fresh re-railing or just a temporary contact problem.

If the decoder still has the driving information, i.e. speed, direction, functions and registration status, stored, it can continue working with this data. Depending on the registration status, it uses either the temporary address assigned via DCC-A or the conventional address. However, if it detects a change in the command station during the next LOGON_ENABLE, the decoder must log in again. If the session ID has changed by more than 1 or has increased by 1 and the operating voltage was interrupted for 3 seconds or longer, the decoder must also log in again. If the controller in the decoder has received an internal reset due to a voltage drop and is therefore unable to measure the time, it can assume that the interruption lasted more than 3 seconds.

If no valid driving information is available and the decoder is on a layout without DCC-A, which can be identified by the non-extended start bit of a DCC packet (see section 1.3), it uses the conventional address.

If the decoder detects a DCC-A system (see section 1.3), it must wait for a LOGON_ENABLE. If the decoder recognizes the command station by its ZID and the session ID hasn't changed, it can continue operation with the known DCC-A address. Otherwise, the decoder must log in again.

If the user programs the conventional address in CV1 or CV17/18 (and CV29 bit 5), the decoder must transfer this to the desired address and delete the command stations and session ID to force a new registration. In addition, bit 7 in the change flags is set (see section 4.2). This bit is also set when the information in Data Room 2 ShortGUI is changed.

Behavior in case of error:

Error conditions during assignment should be displayed to the user.

If the decoder has sent an ID15 response to LOGON_NOW three times (without any other LOGON_* messages in between) and has not been addressed via SELECT or LOGON_ASSIGN, the decoder should assume a failed registration and not start operation. Instead, it should remain stationary with a double-flashing headlight. Recommended flashing pattern: 100ms on, 300ms off, 100ms on, 1500ms off.

If the decoder is assigned the short address 0 during LOGON_ASSIGN, the registration has been recognized by the central unit, but the central unit does not currently want to address the decoder. A different address was previously assigned to the decoder, but the decoder must retain its current state ('Park') except for the motor control. If no other address was previously assigned, the assignment of address 0 is considered an error, and the decoder should indicate this error state (e.g., flashing headlights). In any case, Further registrations with ID15 are not permitted.

7.2 Backoff

If a decoder does not receive confirmation after a login attempt, it will not respond to a certain number of LOGON_ENABLE messages. The number of login requests to be ignored is determined using a random number generated using the unique ID. The first random number is selected from a range of 0 to 7. If the LOGON button is not pressed again, the number is selected from a range of 0 to 15. If the LOGON button is not pressed again, the number is selected from a range of 0 to 31. If the LOGON button is not pressed again, the number is selected from a range of 0 to 63 and will not increment further.

If a LOGON_ENABLE_NOW is received, the decoder ignores the current backoff value and attempts to log in. The backoff value is then removed from the range 0 – 7.

A truly random process must be used for the initial calculation and continuation of the backoff value. If such a process is not available on the decoder,
Alternatively, a pseudorandom process can be used, but care must be taken to ensure that the entropy is sufficiently long (e.g. by repeatedly calculating the unique ID).
In [TN-9.2.1.1] there is a corresponding, easily implementable
algorithm proposal.

Annex A: References to other standards

A.1 Normative references

The standards listed here must be observed in full or within the limits specified in the quotation in order to comply with this standard.

[RCN-211] RCN-211 DCC packet structure, address ranges and global commands

[RCN-212] RCN-212 DCC operating commands for vehicle decoders

[RCN-214] RCN-214 DCC configuration commands

[RCN-217] RCN-217 DCC feedback protocol RailCom

[RCN-225] RCN-225 DCC configuration variables

[TN-218] TN-218 DCC picture and icon numbers for DCC-A

A.2 Informative references

The standards and documents listed here are for information purposes only and are not part of this standard.

[S-9.2.1.1] NMRA: S-9.2.1.1 Advanced Extended Packet Formats

[TN-9.2.1.1] NMRA TN-9.2.1.1 Advanced Extended Packet Formats

[S-9.2.2 Appendix A] NMRA: S-9.2.2 Appendix A DCC Manufacturer ID codes

Appendix B: History

Date	Chapter	Changes from the previous version	version
July 27, 2025	1.5 3.4 / 4.3 4.2 4.2 4.3 5.4 6.1 7.1	CRC and XDCC added to the glossary Filling the data with 0x00 instead of ACK Change flag for changing the conventional address and/or Data Room 2 ShortGUI multiple commands in one package only according to [S-9.2.1.1] All blocks except the last one with 31 bytes of payload Update of the principle symbols, addition of “tram” New registration with changed conventional address If the conventional address is changed, the desired address should also be set accordingly and the central unit and session ID must be deleted	1.1

November 24, 2024	1.2 1.3 1.5 (new) 2.3 3.3 3.3.2 3.3.3 3.3.4 3.4 3.5 3.6 4.2 5 5.2 5.9 6.0 6.0 / 1.5 6.1 6.3 7.1 A.1 E & F	Notation “DCC-A” added Identification of a layout with active DCC-A added Assignment of letters changed, corresponding to CID ÿ ZID Definition of conventional address and DCC-A address Addition of the permanent assignment of an address Addition of storage of the assignment data Evaluation of the RailCom responses to SELECT Changed behavior with ReadBlock Subcommand WriteBlock added Response with ID13 DecoderState Definition of the command sequence Commands SET_DATA and SET_DATA_END added Addition of the permanent assignment of an address Explanation of the terms "temporary" and "permanent" Table of the protocol capabilities of the decoder added Description of data spaces 4 to 7 added Data spaces 0 to 3 mandatory Display of the data spaces without HEADER and CRC Reference to chapter 4.2 corrected Table with packet structure added. Now 16 bytes Manufacturer name, character encoding specified and the format of the linked images defined. Operating mode change only when track voltage is switched off. New definition of when and how the session ID is increased. LOGON_ASSIGN without feedback from the decoder added Chapter added: Detection of interfering decoders Behavior of the decoder when restarting. RCN-211 and RCN-212 added. Annexes E and F added	1.1
November 26, 2023	2 5.4	Reference to TN-218 for the system startup procedure Data space 2, byte 11, bit 4 for data incomplete Meaning of “11” in function information changed	1.1
November 27, 2022	5.6 to 5.9	Data rooms 4 to 7 added. Reference to TN-218 with the image and icon numbers	1.1
September 10, 2021		First version	1.0

Appendix C: Calculation CRC

C.1 Polynom

The polynomial used in this standard represents a compromise between implementation and transmission effort versus data security. Most transmission errors can be sufficiently filtered out by timing analysis of the DCC signal, and the CRC minimizes the residual risk of an error. The 8-bit CRC used here is calculated using the polynomial x8 + x5 + x4 + 1 over the message, starting with the first byte of the message, initialization value = 0 (or the data space number, see section 4.3), not inverted. This value is inserted into the DCC message at the specified location. At the receiver, this calculation is performed over the entire message (without a terminating XOR). If the transmission is error-free, the result is 0, making it easily implementable within the receive logic of a decoder.

Calculating the CRC requires an XOR operation and a table access. The decision as to whether the CRC is correct is based on the first byte of the message and its length, and can therefore also be implemented within the receive logic.

An identical rule is provided in address space 253 (Advanced Extended DCC).

C.2 Calculation example (code optimized)

To simplify the calculation of the CRC, a pre-calculated table is stored in memory:

```
unsigned char crc_array[256] = {
        0x00, 0x5e, 0xbc, 0xe2, 0x61, 0x3f, 0xdd, 0x83,
        0xc2, 0x9c, 0x7e, 0x20, 0xa3, 0xfd, 0x1f, 0x41,
        0x9d, 0xc3, 0x21, 0x7f, 0xfc, 0xa2, 0x40, 0x1e,
        0x5f, 0x01, 0xe3, 0xbd, 0x3e, 0x60, 0x82, 0xdc,
        0x23, 0x7d, 0x9f, 0xc1, 0x42, 0x1c, 0xfe, 0xa0,
        0xe1, 0xbf, 0x5d, 0x03, 0x80, 0xde, 0x3c, 0x62,
        0xbe, 0xe0, 0x02, 0x5c, 0xdf, 0x81, 0x63, 0x3d,
        0x7c, 0x22, 0xc0, 0x9e, 0x1d, 0x43, 0xa1, 0xff,
        0x46, 0x18, 0xfa, 0xa4, 0x27, 0x79, 0x9b, 0xc5,
        0x84, 0xda, 0x38, 0x66, 0xe5, 0xbb, 0x59, 0x07,
        0xdb, 0x85, 0x67, 0x39, 0xba, 0xe4, 0x06, 0x58,
        0x19, 0x47, 0xa5, 0xfb, 0x78, 0x26, 0xc4, 0x9a,
        0x65, 0x3b, 0xd9, 0x87, 0x04, 0x5a, 0xb8, 0xe6,
        0xa7, 0xf9, 0x1b, 0x45, 0xc6, 0x98, 0x7a, 0x24,
        0xf8, 0xa6, 0x44, 0x1a, 0x99, 0xc7, 0x25, 0x7b,
        0x3a, 0x64, 0x86, 0xd8, 0x5b, 0x05, 0xe7, 0xb9,
        0x8c, 0xd2, 0x30, 0x6e, 0xed, 0xb3, 0x51, 0x0f,
        0x4e, 0x10, 0xf2, 0xac, 0x2f, 0x71, 0x93, 0xcd,
        0x11, 0x4f, 0xad, 0xf3, 0x70, 0x2e, 0xcc, 0x92,
        0xd3, 0x8d, 0x6f, 0x31, 0xb2, 0xec, 0x0e, 0x50,
        0xaf, 0xf1, 0x13, 0x4d, 0xce, 0x90, 0x72, 0x2c,
        0x6d, 0x33, 0xd1, 0x8f, 0x0c, 0x52, 0xb0, 0xee,
        0x32, 0x6c, 0x8e, 0xd0, 0x53, 0x0d, 0xef, 0xb1,
        0xf0, 0xae, 0x4c, 0x12, 0x91, 0xcf, 0x2d, 0x73,
        0xca, 0x94, 0x76, 0x28, 0xab, 0xf5, 0x17, 0x49,
        0x08, 0x56, 0xb4, 0xea, 0x69, 0x37, 0xd5, 0x8b,
        0x57, 0x09, 0xeb, 0xb5, 0x36, 0x68, 0x8a, 0xd4,
        0x95, 0xcb, 0x29, 0x77, 0xf4, 0xaa, 0x48, 0x16,
        0xe9, 0xb7, 0x55, 0x0b, 0x88, 0xd6, 0x34, 0x6a,
        0x2b, 0x75, 0x97, 0xc9, 0x4a, 0x14, 0xf6, 0xa8,
        0x74, 0x2a, 0xc8, 0x96, 0x15, 0x4b, 0xa9, 0xf7,
        0xb6, 0xe8, 0x0a, 0x54, 0xd7, 0x89, 0x6b, 0x35,
        };
```

Now you can simplify 8 division steps into one table access:

```
crc_value = crc_array[message[i] ^ crc_value];
```

The calculation at the headquarters is then as follows:

```
crc_value = 0;
for (i=0; i<sizeof(DCC_payload); i++) {
        crc_value = crc_array[DCC_payload[i] ^ } crc_value];
```

The check in the decoder is carried out analogously:

```
crc_value = 0;
for (i=0; i<sizeof(DCC_with_CRC); i++) {
        crc_value = crc_array[DCC_payload[i] ^ } crc_value];

if (crc_value != 0) error();
```

C.3 Calculation example (memory space optimized)

If memory space on the implementation platform is the limiting resource, the CRC can also be calculated without a table. An example implementation using a bit query:

```
unsigned char crc_calc(unsigned char data) {
        unsigned char result = 0;

        if(data & 1) result ^= 0x5e;
        if(data & 2) result ^= 0xbc;
        if(data & 4) result ^= 0x61;
        if(data & 8) result ^= 0xc2;
        if(data & 0x10) result ^= 0x9d;
        if(data & 0x20) result ^= 0x23;
        if(data & 0x40) result ^= 0x46;
        if(data & 0x80) result ^= 0x8c;
        return result;
}
```

The calculation then results in:

```
crc_value = 0;
for (i=0; i<sizeof(DCC_payload); i++) {
        crc_value = crc_calc(DCC_payload[i] ^ } crc_value];
```

C.4 Calculation example (size and code optimized)

As a third example, a division into table methods (with smaller tables) and short Code:

```
unsigned char crc_nibble1[16] = {
        0x00, 0x5e, 0xbc, 0xe2, 0x61, 0x3f, 0xdd, 0x83, 0xc2, 0x9c, 0x7e, 0x20, 0xa3,
        0xfd, 0x1f, 0x41,
        };

unsigned char crcnibble2[16] = {
        0x00, 0x9d, 0x23, 0xbe, 0x46, 0xdb, 0x65, 0xf8,
        0x8c, 0x11, 0xaf, 0x32, 0xca, 0x57, 0xe9, 0x74
};
```

```
unsigned char crc_calc(unsigned char data) {
        unsigned char result = 0;

        result = crc_nibble1[data&0xf] ^ crc_nibble2[data>>4];
        return result;
}
```

C.5 Sample data

For (simplified) implementation control, here is a numerical example:

0x0B 0x0A 0x00 0x00 0x8E 0x40 0x00 0x0D 0x67 0x00 0x01 0x00 results in a CRC byte 0x4C.

Appendix D: Addresses

When transmitting the desired address and when assigning the address using LOGON_ASSIGN, the (historically grown) addresses are mapped to 14 bits.

This assignment is made as follows: (A13-A8, A7-A0)

A13.. A8 decoders		Description
0..0x27 (0.. 39)	Vehicle decoder, long address	Address results in A13...A0 CV17 = A13...A8 + 0xC0 CV18 = A7...A0
0x28..0x2F (40..47)	Extended Accessory decoder	The address is A10...A0; this is the linear address according to RCN213.
0x30..0x37 (48..55)	standard Accessory decoder	The address is A10...A0; this is the linear address according to RCN213.
0x38	Vehicle decoder, short address	Address results in A6...A0 (corresponds to CV1)
0x39..0x3E -		reserved
0x3F	-	Decoder is in FW update mode, only firmware update possible

Informative: For mobile decoders, the transition from short to long address of 127 to 128. In the high-order byte, the upper two bits are reserved and set to 1. The transmission of a short address, for example, becomes 0xF8 in the high-order byte and the short address in the low-order byte.

Appendix E: Calculation example for the ZID

It is advisable to use the available 16-bit values for the ZID as evenly distributed as possible in order to minimize the probability of false detection by a control center.
The recommended, low-effort approach is to calculate a CRC8 (with start value 0) from the bytes of the full central identification (manufacturer, product/serial number) using the bytes with odd index (1, 3 ...) and to use this as the low byte for the ZID.
Similarly, the CRC is calculated over the bytes with even index (2, 4 ...) and used the as the high-order byte for the ZID. If both results equal 0 (which corresponds to a reserved value of the ZID), the starting value is incremented by 1 and the calculation is repeated.

Appendix F: Example for describing a namespace

Preliminary note: In this example, all **bytes** are represented in hexadecimal format without a "0x" prefix. The abbreviation CRC (lowercase) denotes the cyclic redundancy checksum of the DCC command, and CRC (uppercase) denotes the checksum for the write operation.

Given is the decoder with the UID **0 0D AF FE D0 0F,** where the first 12 bits **0 0D** are the manufacturer identifier and the following 16 bits **AF FE D0 0F** are the product identifier.
There, data space 5 (long name) is to be described with the text "abcdefghijk\0". \0 is the terminating 0, so the text has a total length of 12 bytes.

The command sequence from the control center is:

1. SELECT command

FE D0 0D AF FE D0 0F FC 05 00 0C crc xor

Explanation:

FE	= DCC-A command
D0 0D	= SELECT (0xD) and manufacturer ID (0x00D)
AF FE D0 0F	= Product identifier
FC	= WriteBlock subcommand
05	= Number of the data room
00 0C	= Size
crc	= Checksum of this command according to chapter 1.4. (A)
xor	= DCC check byte ([RCN-211] Chapter 2)

2. first SET_DATA command

FE 02 61 62 63 64 65 66 67 68 69 6a 6b crc xor

Explanation:

FE	= DCC-A command
02	= SET_DATA
61 … **6b**	= payload data (here 11 bytes)
crc	= Checksum of this command according to chapter 1.4.

xor = DCC check byte

3. second SET_DATA command

FE 02 00 CRC xor

Explanation:

FE = DCC-A command
02 = SET_DATA
00 = Payload data
CRC = Checksum of the write process
xor = DCC check byte

4. SET_DATA_END command

FE 03 xor

The DCC checksum (A) passed during the SELECT is used as the starting value for the further calculation of the checksum for the write operation. In the following SET_DATA commands, this CRC is then further calculated with the payload data (61, 62, ...) and then appended to the last byte of the payload data (here, 0) in the second SET_DATA command.

The total checksum is calculated using:

FE D0 0D AF FE D0 0F FC 05 00 0C 61 62 63 64 65 66 67 68 69 6a 6b 00

The second SET_DATA is short (less than or equal to 6 bytes). Therefore, no local checksum "crc" is appended.
