railDECOrc

RailCom transmitter & function decoder
User Manual

Firmware v1.0.0 · PIC12F629 only
Document revision 1.0 · 14 November 2026 · English

> **Before programming the PIC:** back up its individual oscillator calibration at `0x03FF`. Program the complete release image, including EEPROM. Never replace the calibration with a value from another PIC.

railDECOrc adds locomotive address feedback and three configurable lighting outputs to a DCC-equipped model. It can work alongside an existing locomotive decoder, including one without RailCom.

Project website: tren.enmicasa.net — RailPIC projects and model railway electronics.

Contents

1. Overview & operating principle
2. Connections & installation
3. Programming the firmware
4. First run & everyday operation
5. CV programming & addressing
6. CV reference: standard settings
7. CV reference: secondary address & outputs
8. Configuration examples
9. Troubleshooting & recovery
10. Acknowledgements & references

1. Overview & operating principle

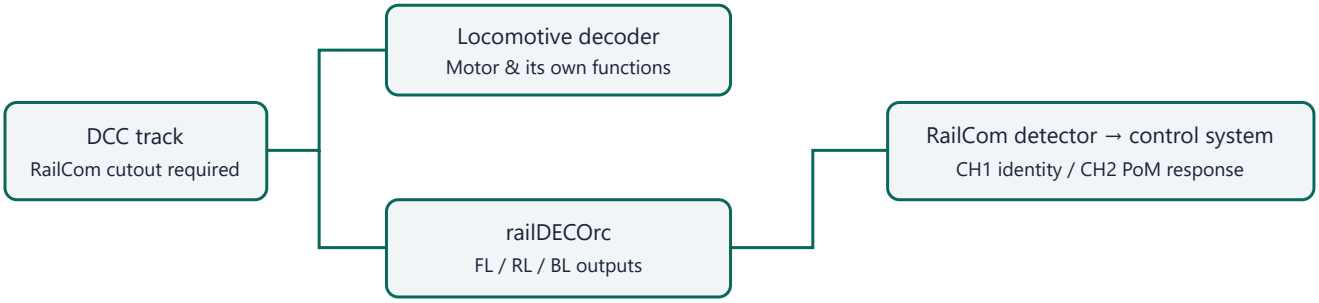
railDECOrc is a small DCC function decoder and RailCom transmitter based on the PIC12F629. It does not drive a motor. When installed alongside a locomotive decoder, both devices receive the track signal in parallel. The locomotive decoder continues to control the motor; railDECOrc handles its own outputs and feedback.

Locomotive identity

The **primary address** receives speed, direction and function commands and is broadcast on RailCom channel 1 (CH1). Normally, set it to the same address as the locomotive decoder.

Independent configuration

The **secondary address** is dedicated to Programming on the Main (PoM). It allows railDECOrc to be configured without sending those CV changes to the locomotive decoder.



What this release provides

- Short and long primary addresses; a separately configurable short or long PoM address.
- Three outputs, each mapped to **F0–F68**, with a direction filter and either steady light or braking sparks.
- PoM byte reads and writes of railDECOrc's own CVs. Reads use RailCom CH2.
- Direct-mode byte writes for service-mode recovery.

> **Feedback is not a proxy for the motor decoder.** A PoM read from railDECOrc returns its own stored CV value. It cannot read the CVs inside the other decoder.

RailCom needs a command station/booster that generates the cutout and a compatible detector. CH1 identification and CH2 CV replies are separate capabilities: a detector that displays addresses does not necessarily support PoM readback.

This release is DCC-only: there is no analogue/DC operating mode. PIC12F675 and PIC12F683 are not supported by the v1.0.0 image.

2. Connections & installation

> **Switch off track power before wiring.** Keep the board insulated from the chassis, motor terminals and other electronics. Do not attach a PIC programmer while the model is connected to powered track.

Use the schematic and connection labels for your railDECOrc board to identify the track inputs, lighting outputs and local ground before wiring.

Connection	Purpose	Installation notes
TrkA / TrkB TRACK_r / TRACK_n	DCC track inputs	Connect to the two rail pickups, in parallel with the locomotive decoder's track inputs. Do not connect to its motor outputs.
FL	Output on GP5, PIC pin 2	Traditionally front light; configurable. Factory assignment: F29.
RL	Output on GP4, PIC pin 3	Traditionally rear light; configurable. Factory assignment: F30.
BL / FB	Output on GP0, PIC pin 7	Brake-light output; labelled FB on the RailPIC schematic. Factory assignment: F31, steady light.
GND / VSS	Local circuit ground	Reference for LED outputs. This is not either rail, a motor terminal or automatically the same potential as another decoder's common wire.
CAP, if provided	Optional capacitor connection	Only use the provision documented for your board. Observe polarity and voltage rating. CAP is not a universal connection available on every PCB.

LED wiring

The outputs are PIC logic outputs, not high-current accessory drivers. Connect the LED from the output through a series resistor to local GND, with its anode towards the output. Every LED requires a current-limiting resistor. Start with a low-current LED load; never connect a lamp, motor, relay or unresisted LED directly.

Do not exceed **20 mA per output**. Treat that as a ceiling, not a recommended LED current or permission to draw 60 mA continuously across all outputs. Check the PIC's combined current limits and regulator dissipation. A few milliamperes is usually sufficient for model lighting.

For an approximate 5 V output and a white LED with a 3 V forward drop, a 1 kΩ series resistor gives roughly 2 mA: `R = (5 – 3) / 0.002 = 1000 Ω`. Check polarity and brightness before installation.

Before applying power

1. Check component orientation, solder bridges, insulation and the correct PIC type.
2. Check that each LED has its own resistor and that no output is tied to another decoder's output or common-positive wire.
3. Verify track voltage is suitable for the actual components and regulator fitted.
4. Test on a protected, current-limited DCC output before closing the model.

The RailPIC hardware schematic provides circuit details. Check the dimensions and pad locations on your actual board before mounting it.

3. Programming the firmware

Flashing the PIC installs executable code and its initial configuration. It is different from changing CVs over DCC. Use a programmer and software that explicitly support the **PIC12F629** and preserve its factory oscillator calibration.

Release image

```
raildecoRC_v1.0.0_pic12f629.hex
```

This is the complete, frozen v1.0.0 release image, containing program, configuration and EEPROM data. Its SHA-256 checksum is:

```
574e023136d5d95bce4638d03ec4c2161baa5dbec344ef216bc977b409e64ccc
```

On Windows, check the downloaded file using PowerShell:

```
Get-FileHash .\raildecoRC_v1.0.0_pic12f629.hex -Algorithm SHA256
```

Programming procedure

1. **Disconnect all power.** Prefer programming the PIC in a suitable socket/adaptor outside the board. In-circuit programming is only appropriate after checking the board's loading, power and programming-voltage paths; a supported ICSP header is not assumed.
2. Select **PIC12F629** in the programmer software. Read the device before erasing it. Save a backup and record the calibration instruction at program-memory word address **`0x03FF`**.
3. Confirm the programmer's calibration-preservation procedure. The value is individual to each PIC. If it is missing or invalid, stop: loading this HEX will not recreate the factory calibration.
4. Load the release HEX. Enable programming of **program memory, configuration and data EEPROM**. Use the configuration supplied by the image (`0x3FC4`); do not invent alternative fuse settings.
5. Erase/program using the programmer's documented sequence, preserving or restoring the recorded calibration. Never continue if the software warns that it will erase OSCCAL without restoring it.
6. Run verification. Read back the PIC and check the calibration at `0x03FF` is unchanged. Verify program, configuration and all 128 EEPROM bytes against the image, allowing for the device-specific calibration word, which the release deliberately does not supply.
7. Disconnect the programmer, install the PIC in the correct orientation and perform the first-run checks in the next chapter.

> **Program firmware and EEPROM together.** Use the complete release image so that the program and its configuration data match. An intermediate build file ending in `_assembled.hex` is not a substitute for the complete release image.

Programming connections, if using a verified adaptor

PIC12F629: pin 1 VDD, pin 8 VSS, pin 4 VPP/MCLR, pin 7 ICSPDAT (GP0), pin 6 ICSPCLK (GP1). These pins also connect to the operating circuit; this list is not approval to connect a programmer directly to an assembled board. Follow the programmer and PIC documentation for supply and programming voltages.

The filename version is **v1.0.0**. The tested image intentionally retains EEPROM identification `v1.0` and CV7 = **11**; these are not an indication that the wrong release was loaded.

4. First run & everyday operation

Factory configuration

Setting	Factory value
Primary / operating address	Short address **3**
Secondary / PoM address	Long address **9000**
FL / RL / BL function	**F29 / F30 / F31**
Direction filters / effects	Both directions / steady light on all three outputs
RailCom	Enabled; CH1 and CH2 enabled (CV28 = 3, CV29 = 10)
Speed-step setting	28/128 steps (CV29 bit 1 = 1)

1. Place the model on a protected DCC track. Initially test railDECOrc alone if other decoders could respond to the same configuration commands.
2. Enable RailCom cutouts in the command station/booster and use a detector suited to the feedback you want to observe.
3. Select locomotive **short address 3**. Send F29, F30 and F31 on and off to check FL, RL and BL respectively. Your control software must actually transmit these high functions.
4. Check the detector reports primary address 3 on CH1.
5. For a CV check, send a **PoM byte read to long address 9000**, for example CV7 (11) or CV8 (13). This requires a working CH2 readback path.
6. Set the primary address to match the locomotive decoder. Reserve a suitable, preferably unique secondary address for this module.

> **9000 must be a long address.** Secondary address 9000 is for CV programming, not for switching the lights. Send output functions to the primary address.

Outputs and braking sparks

For a steady output, the mapped function switches the LED on or off, subject to its direction filter. With the sparks effect selected, the mapped function *enables* the braking effect: it does not force the output on continuously. A reduction in commanded speed triggers a temporary pseudo-random flicker.

All outputs configured for sparks share the braking window and flicker pattern, but each keeps its own function enable and direction filter. The effect responds to DCC speed commands, not to a physical speed sensor.

State changes

Outputs start off at power-up. Accepted configuration writes reload the settings and clear the remembered function states, so the command station must refresh the functions afterwards. If DCC packets stop, function states are retained while power remains available; any active spark effect ends when its timer expires. There is no transition to analogue operation.

Do not use a function output as a safety interlock: packet loss does not guarantee that a steady output turns off.

5. CV programming & addressing

Primary versus secondary

The primary address is stored in CV1 or CV17/CV18 and selected by CV29 bit 5. It handles locomotive commands and the CH1 address announcement. The secondary address is configured through CV112–115 and handles PoM reads and writes. It does not replace the primary address in CH1 and does not control the outputs.

With the secondary address active, railDECOrc ignores PoM addressed to the primary address. This lets you program the motor decoder at the locomotive address and railDECOrc at its own secondary address. Use a secondary address that no other installed decoder uses for PoM. Deliberately identical primary and secondary addresses are allowed, but lose this separation.

> **Service-mode writes are not isolated by address.** If two decoders are connected to the programming track, both may accept a write. Disconnect the other decoder before service-mode recovery or configuration.

Supported programming methods

Method	v1.0.0 behaviour
PoM byte write	Supported at the secondary address. The command station must repeat the identical write packet; the firmware requires repetition before applying it.
PoM byte read	Supported at the secondary address, with reply on RailCom CH2. Returns railDECOrc's own CVs only.
Direct service-mode byte write	Supported after DCC reset packets; useful for recovery. No service-mode current acknowledgement is generated.
Paged / bit programming	Not implemented. PoM bit manipulation and Direct bit programming are not supported.
Service-mode read / ACK	No current ACK readback. A command station may report a failure even when a Direct byte write was applied.

Enter the public CV numbers shown in this manual, such as **CV112**, into the command station. Do not substitute a zero-based packet index or a physical EEPROM address.

Changing the secondary address safely

1. At the **current** secondary address, write CV112 for a new short address, or CV113 and CV114 for a new long address.
2. These values are only prepared; the old PoM address remains active.
3. Still using the old address, write **CV115 = 1** to apply the prepared short address, or **CV115 = 2** to apply the prepared long address.
4. Send subsequent PoM operations to the **new** secondary address, with the correct short/long type. Check it using a CV read if CH2 is available.

Long-address calculation

For an address `A`, use `high = 192 + floor(A / 256)` and `low = A mod 256`. For the primary address, write high/low to CV17/CV18 and set CV29 bit 5. For the secondary address, write them to CV113/CV114 and apply with CV115 = 2.

Use addresses 1–10239 for long addressing (high byte 192–231). Short addresses are 1–127. Short and long address types are distinct even when their numerical values are equal.

6. CV reference: standard settings

Values are decimal unless prefixed by `0x`. Defaults below describe the complete v1.0.0 release image and a factory reset.

CV	Default	Setting / accepted values
1	3	Primary short address, 1–127. Used when CV29 bit 5 is 0.
7	11	Firmware revision identifier, read-only. Release version is v1.0.0.
8	13	DIY decoder manufacturer identifier, read-only. Writing **33** requests a factory reset; it does not change the identifier.
17	192	Primary long-address high byte, 192–231.
18	100	Primary long-address low byte, 0–255. The stored factory long address is 100, but it is not selected initially.
28	3	RailCom channel selection. Bit 0 = CH1 address broadcast; bit 1 = CH2 data replies. Use 0–3 for these defined bits.
29	10	Decoder configuration; defined bits below. Analogue bit 2 is always cleared by firmware.

CV29 bits

Bit	Weight	0	1
0	1	Normal direction	Reversed direction
1	2	14 speed steps	28/128 speed steps
2	4	DCC only	Not supported; forced to 0
3	8	RailCom disabled	RailCom enabled
5	32	Primary short address (CV1)	Primary long address (CV17/CV18)

Leave unused bits 4, 6 and 7 at zero. Add the weights of the desired enabled bits. Typical settings:

- **10 = 2 + 8**: normal direction, 28/128 steps, RailCom, primary short address.
- **42 = 2 + 8 + 32**: the same, with primary long address.
- **11 / 43**: corresponding settings with reversed direction.

Direction filters use the direction after applying CV29 bit 0. In 14-step mode, F0 is taken from the speed packet; match this setting to the command station.

RailCom enable conditions

Set CV29 bit 3 to enable RailCom globally. CV28 bit 0 enables CH1; bit 1 enables CH2. For both channels use CV28 = 3. Disabling CH2 prevents PoM readback even if CH1 address detection still works. Disabling RailCom does not itself remove the ability to send PoM writes.

Unsupported CV reads return zero, and unsupported writes are ignored. Only the CVs documented in this manual are supported; railDECOrc does not mirror the CVs of another decoder.

7. CV reference: secondary address & outputs

Secondary PoM address

CV	Default	Meaning
112	127	Prepared short secondary address, 1–127. Does not activate it.
113	227	Prepared long secondary high byte, 192–231.
114	40	Prepared long secondary low byte, 0–255. Defaults 227/40 encode 9000.
115	2	Write 1 to activate the prepared short address; write 2 to activate the prepared long address. Reading reports the active type (1 or 2).

Reading CV112–114 shows the prepared values, which may differ from the active address if they have been edited but not applied. CV115 is an apply command/type indicator, not a RailCom enable switch. Other values are ignored.

Output mapping

Output	Function CV	Direction CV	Effect CV
FL / GP5	116 = **29**	117 = **0**	118 = **0**
RL / GP4	119 = **30**	120 = **0**	121 = **0**
BL (FB) / GP0	122 = **31**	123 = **0**	124 = **0**

Field	Accepted values
Function	**0–68** maps F0–F68; **255** disables the output.
Direction	**0** = both directions; **1** = forward only; **2** = reverse only.
Effect	**0** = steady light controlled by the function; **1** = braking sparks enabled by the function.

Invalid mapping values are ignored. Function numbers are not bit masks: write **29** to select F29, not a power of two. The three outputs can share a function or use independent functions.

Practical implications

- F29/F30/F31 are factory choices, not a limit. F0 through F68 are supported by firmware.
- A controller must generate the appropriate DCC function groups. An on-screen function button alone does not prove the packet was sent.
- For braking sparks, keep the mapped function on and command a reduction in speed at the primary address. No speed reduction means no new braking trigger.
- Changing a CV clears function latches. Refresh the functions after completing a group of configuration writes.

> **Remember the destination:** write mapping CVs at the secondary PoM address; operate the mapped functions at the primary address.

8. Configuration examples

Examples assume factory secondary **long address 9000**. If you have changed it, use its current address and type instead. Select PoM **byte write**. After configuration, refresh the function states at the primary address.

A. Match primary short address 24

```
PoM destination: long 9000
CV1  = 24
CV29 = 10

Operate functions at: short 24
Continue configuring railDECOrc at: long 9000
```

Set the motor decoder to address 24 separately, using its own instructions. CV29 = 10 assumes normal direction, 28/128 steps and RailCom enabled.

B. Use primary long address 1234

```
1234 = 4 × 256 + 210 → high = 192 + 4 = 196
PoM destination: long 9000
CV17 = 196
CV18 = 210
CV29 = 42

Operate functions at: long 1234
```

C. Restore conventional directional F0 lighting

```
PoM destination: long 9000
CV116 = 0     FL uses F0
CV117 = 1     FL forward only
CV118 = 0     FL steady
CV119 = 0     RL uses F0
CV120 = 2     RL reverse only
CV121 = 0     RL steady
```

Switch F0 on at the primary address and change direction. If the lights are reversed relative to the model, check wiring and the intended CV29 direction setting.

D. Enable BL braking sparks with F31

```
PoM destination: long 9000
CV122 = 31
CV123 = 0
CV124 = 1
```

At the primary address, turn F31 on, command a running speed and then reduce it. Turn F31 off to inhibit sparks. To return to steady BL, write CV124 = 0. To disable BL entirely, write CV122 = 255.

E. Move secondary PoM address to long 9100

```
PoM destination: long 9000
CV113 = 227
CV114 = 140
CV115 = 2     Apply last

New PoM destination: long 9100
```

Alternatively, for short secondary address 120: at the current address write CV112 = 120, then CV115 = 1. Continue PoM at **short 120**. Reserve these example addresses before using them on a live layout.

9. Troubleshooting & recovery

Symptom	Checks
No output responds	Check track power, PIC orientation, firmware/EEPROM verification, LED polarity and resistors. Send functions to the primary address, not the secondary. Factory outputs use F29–F31, not F0.
Low functions work, high functions do not	Verify the controller actually sends the required F29–F68 group. Check the mapping CV and direction filter. Resend the function after programming.
CV writes appear ineffective	Use the current secondary address and correct short/long type. Select byte writes; PoM writes require repeated identical packets. Check that values are in range. Function latches are cleared by configuration writes.
CH1 address is missing	Check the RailCom cutout, detector wiring, CV29 bit 3 and CV28 bit 0. The broadcast address is the primary, not the secondary.
CH1 works but CV reads fail	Check CV28 bit 1 and the complete CH2 receiver/readback path. Confirm that the detector and command station support CH2 PoM replies, not only CH1 address identification.
BL stays off in sparks mode	Enable its mapped function, check direction filtering, then command a speed reduction at the primary address. This effect is not an always-on flicker.
Service-mode write reports an error	There is no current ACK. The write may still have applied. Use Direct byte writes after reset packets; verify later by PoM read if available. Paged and bit modes will not work.
Feedback or timing fails after flashing	Recheck the device selection, release checksum, configuration and individual OSCCAL. Do not try random calibration values.

Factory reset at a known secondary address

1. Send a PoM byte write of **33 to CV8** at the current secondary address.
2. The reset restores primary short address **3**, secondary long address **9000**, factory mappings F29/F30/F31 and the other documented defaults.
3. Use **long 9000** for the next PoM command. CV8 still reads 13; the identification string is retained.

If the secondary address is unknown

Isolate railDECOrc from every other decoder. Use a service programming output to send DCC reset packets followed by a **Direct-mode byte write, CV8 = 33**. The command station must support that sequence without requiring a current ACK to complete the write. Do not repeat blindly with another decoder connected.

If the command station cannot perform this recovery, use a suitable PIC programmer to reinstall the complete release image, including EEPROM, following chapter 3. This also restores factory settings. Preserve the PIC's individual OSCCAL again.

> **Do not diagnose a possible short circuit by repeated power cycling.** Remove power, inspect the wiring and measure with appropriate tools before trying again.

Record your settings

Keep a note of both address types and values, the three mapping triples and CV28/CV29. Prepared CV112–114 values alone may not identify the currently active secondary address.

10. Acknowledgements & references

A community effort over many years

RailPIC warmly thanks all the railway modellers who have helped over the years: those who built and tested boards, tried firmware on their layouts, reported problems, shared ideas and practical experience, and offered patient advice and encouragement. Their time, curiosity and generosity have helped these projects grow.

Special thanks go to Paco Cañadas, creator of DCC-Rail. His work is the foundation and inspiration for railDECOrc. The hardware circuit used here is the same as his original DCC-Rail design; RailPIC does not claim authorship of that circuit. The board implementation and the firmware extensions presented in this manual build on that contribution.

The firmware also has a direct technical lineage: the original firmware was reconstructed into an equivalent ASM baseline before the secondary PoM address and configurable F0–F68 mappings were added. It would therefore be inaccurate to describe railDECOrc as an entirely independent design merely inspired by the original.

Thank you to everyone who has contributed, whether through a detailed test report, a suggestion, a conversation or the simple willingness to try an early version on a real model railway.

Original project and technical references

- Paco's model railway electronics website — original DCC-Rail project. The historical manual credits its author as F. Cañada.
- RailPIC hardware schematic: raildecoRC.pdf. Check the schematic and component documentation for your specific board.
- RailPIC firmware release: `raildecoRC_v1.0.0_pic12f629.hex`; maintained source: `raildecoRC.asm`.
- Microchip PIC12F629 documentation — electrical limits, programming specification and calibration requirements.
- NMRA S-9.2.1, 24 January 2025 — extended packet formats, including function groups through F68.
- NMRA S-9.2.2, October 2012 — CV definitions and manufacturer-use range. CV112–124 are in that range.

Release scope and evidence

This manual describes the PIC12F629 v1.0.0 release, not historical firmware or proposed future devices. The project maintainer has confirmed this version works on hardware. Host tests and a byte-identical rebuild support the release record, but do not constitute independent electrical testing or validation of every layout/controller combination. No NMRA certification is claimed.

The braking effect preserves the original operating concept, but uses a different pseudo-random generator; identical flash sequences are not claimed. Historical analogue, Paged and other programming features must not be assumed to exist in this release.

Attribution and publication

Attribution does not itself grant permission to redistribute third-party material. This manual links to the original project rather than reproducing its manual, photographs or PCB artwork, and does not assign a new licence to that material. Any redistribution remains subject to the applicable permissions.
