DCC EX Command Reference

Edit New Page

Fred edited this page 13 minutes ago · 13 revisions

DCC++ EX Command Reference

DCC++ EX Provides an Application Programming Interface (API) that other applications use to send simple text commands that can operate your Command Station. Several "front end" controllers are available or you can easily create your own. Here are some examples:

- exWebThrottle - Our DCC++ EX browser based throttle using your USB cable

  See it and run from the web here

  Download and run from your computer

- Engine Driver - Cellphone App WiFi Throttle
- The Command Line from a Serial Monitor like the Arduino Serial Monitor or Putty
- DCC++ Controller Software
- JMRI Railroad ControlSofware

This reference explains the available command structure, and for commands that provide them, their responses. If you are testing your Command Station or writing your own control program, make sure you have the latest release of the DCC++ EX Command Station Firmware.

Pages 3

Find a Page...

Home

DCC EX Command Reference

Introduction to DCC

+ Add a custom sideba

Clone this wiki locally

https://github.com/DCC-

You can view and edit this code in the Arduino IDE or in PlatformIO Software from GitHub. If you are new to we suggest you star with the DCC++ EX Webpage.

SINGLE LETTER COMMANDS

- **<s> Lower Case "s"**: DCC++ EX CommandStation Status
  Returns: Track power status,Throttle status, Turn-out status, and a version number.

- **<0> Number Zero**: Turn Power **OFF** to tracks (Both Main & Programming).
  Returns: **<p0>** Power to tracks OFF. (See extended power command below)

- **<1> Number One**: Turn Power **ON** to tracks (Both Main & Programming).
  Returns: **<p1>**: Power to tracks ON. (See extended power command below)

- **<T> Upper Case T**: Lists all defined turnouts.
  Returns: **<H ID ADDRESS SUBADDRESS THROW>** for each defined turnout or **<X>** if no turnouts defined.
  ID - ID assigned to the turnout
  ADDRESS, SUBADDRESS - The two part address of the turnout. See this formula for how the address,subaddress pair is calculated
  THROW - False or a "0" is unthrown. True or "1" is thrown.

- **<S> Upper Case S**: Lists all defined sensors.
  Returns: **<Q ID PIN PULLUP>** for each defined sensor or **<X>** if no sensors defined.

- **<Z> Upper Case Z**: Lists all defined output pins
  Returns: **<Y ID PIN IFLAG STATE>** for each defined output pin or **<X>** if no output pins defined

- **<Q> Upper Case Q**: Lists Status of all sensors.
  Returns: **<Q ID>** (active) or **<q ID>** (not active)

- **<E> Upper case E**: Command to **Store** definitions to EEPROM
  Returns: **<e nTurnouts nSensors&gt**

- **<c> Lower case c**: Displays the instantaneous current on the MAIN Track
  Returns: **<c CURRENT>**, where CURRENT is the Raw value of the current sense pin

- **<e> Lower Case e**: Command to **Erase ALL (turnouts, sensors, and outputs)** definitions from EEPROM
  Returns: **<0>** EEPROM Empty

  (NOTE:There is NO Un-Delete)

- **<D> Upper Case D**: Please See Diagnostics-<D>-Command Page

- There are a few other Debugging commands in SerialCommand.cpp that should only be used by advanced users (Potentially Harmful if not used correctly).

Track Power Commands

<0|1> - Turns power to both tracks on or off

examples:

<1> - Turn power to all tracks on

<0> - Turn power to all tracks off

<0|1 MAIN|PROG|JOIN> - Turns power on and off to the MAIN and PROG tracks independently from each other and allows joining the MAIN and PROG tracks together

Examples:

<1 MAIN> - Turns on power just to the MAIN track

<0 PROG> - Turns off power just to the PROG track

<1 JOIN> - Joins both tracks together to be both MAIN

Engine Decoder (CAB) Operation Commands

THROTTLE

The CAB throttle format is **<t REGISTER CAB SPEED DIRECTION>**.

Breakdown for this example **<t 1 03 20 1>** is:
"<" = Start delimiter of a DCC++ EX command. (A space after **<** is not required but acceptable)
"t" = (lower case t) This command is for a Decoder installed in a engine or simply a "cab".
"1" = deprecated. We no longer use this but need something here for compatibility with legacy systems. Enter any number value. **"03"** = CAB: the short (1-127) or long (128-10293) address of the engine decoder (this has to be already programmed in the decoder) See Programming Commands bellow.
"20" = SPEED: throttle speed from 0-126, or -1 for emergency stop (resets SPEED to 0)
"1" = DIRECTION: 1=forward, 0=reverse. Setting direction when speed=0 or speed=-1 only effects directionality of cab lighting for a stopped train
">" = I am the end of this command

If the command was successful the serial monitor should reply with : **<T 1 20 1>** meaning :
"<" = Begin DCC++ EX command
"T" = (upper case T) DCC++ EX Cab command was sent from DCC++ EX Command Station **"1"** = register 1 was changed
"20" = set to speed 20
"1" = forward direction
">" = End DCC++ EX command

CAB FUNCTIONS

There are two formats for setting CAB functions, the DCC++ Classic legacy method (maintained for compatibility) and the new DCC++ EX method. Both methods are described here though new applications are encouraged to use the newer command (capital F vs. small f).

- This turns on and off engine decoder functions

- F0-F28 (F0 is sometimes called FL)
- NOTE: setting requests are transmitted directly to mobile engine decoder
- current state of engine functions is not stored by the DCC++ CommandStation
- All functions Groups get set all at once per NMRA DCC standards

CAB Functions format is **<F CAB FUNC 1|0>**

To set functions F0-F28 on=(1) or off=(0): <F CAB FUNC 0|1>

- **"<"** = Begin DCC++ EX command
- **"F"** = (upper case F) This command is for a CAB function ie: Lights, horn, bell
- CAB: the short (1-127) or long (128-10293) address of the engine decoder
- FUNC: the CAB function number (0-28) whose function is defined by your decoder
- 0|1: a value of 0 to set the function OFF and 1 to set the function ON
- **">"** = End DCC++ EX command

Examples: <F 3 0 1> Turns the headlight ON for CAB (loco address) 3 <F 126 0 0> Turns the headlight OFF for CAB 126 <F 1330 1 1> Turns the horn ON for CAB 1330

The Legacy CAB Functions format is **<f CAB BYTE1 [BYTE2]>**

To set functions F0-F4 on=(1) or off=(0): <f CAB BYTE1 [BYTE2]>

- < = Begin DCC++ EX command
- f = (lower case f) This command is for a CAB function ie: Lights, horn, bell
- CAB: the short (1-127) or long (128-10293) address of the engine decoder
- BYTE1: 128 + F1*1 + F2*2 + F3*4 + F4*8 + F0*16
- ADD the ones you want **ON** together
- Add 1 for F1 ON
- Add 2 for F2 ON

- Add 4 for F3 ON
- Add 8 for F4 ON
- Add 16 for F0 ON
- 128 Alone Turns OFF **F0-F4**
- BYTE2: omitted
- > = End DCC++ EX command

> To make BYTE1 add the values of what you want ON together,
> the ones that you want OFF do not get added to the base value of 128.
> F0 (Light)=16, F1 (Bell)=1, F2 (Horn)=2, F3=4, F4=8
> All off = 128
> Light on 128 + 16 = 144
> Light and bell on 128 + 16 + 1 = 145
> Light and horn on 128 + 16 + 2 = 146
> Just horn 128 + 2 = 130

> If light is on (144), Then you turn on bell with light (145), Bell back off but light on (144)

Breakdown for this example **<f 3265 144>**
"<" = Begin DCC++ EX command
"f" = (lower case f) This command is for a CAB,s function ie: Lights, horn, bell
"3265" = CAB: the short (1-127) or long (128-10293) address of the engine decoder
"144" = Turn on headlight
">" = End DCC++ EX command

To set functions F5-F8 on=(1) or off=(0): <f CAB BYTE1 [BYTE2]>

- "<" = Begin DCC++ EX command
- "f" = (lower case f) This command is for a CAB,s function.
- BYTE1: 176 + F5*1 + F6*2 + F7*4 + F8*8
- ADD 176 + the ones you want **ON** together
- Add 1 for F5 ON
- Add 2 for F6 ON
- Add 4 for F7 ON

- Add 8 for F8 ON
- 176 Alone Turns OFF **F5-F8**
- BYTE2: omitted
- ">" = End DCC++ EX command

To set functions F9-F12 on=(1) or off=(0): <f CAB BYTE1 [BYTE2]>

- "<" = Begin DCC++ EX command
- "f" = (lower case f) This command is for a CAB,s function.
- BYTE1: 160 + F9*1 +F10*2 + F11*4 + F12*8
- ADD 160 + the ones you want **ON** together
- Add 1 for F9 ON
- Add 2 for F10 ON
- Add 4 for F11 ON
- Add 8 for F12 ON
- 160 Alone Turns OFF **F9-F12**
- BYTE2: omitted
- ">" = End DCC++ EX command

To set functions F13-F20 on=(1) or off=(0): <f CAB BYTE1 [BYTE2]>

- "<" = Begin DCC++ EX command
- "f" = (lower case f) This command is for a CAB,s function.
- BYTE1: 222
- BYTE2: F13*1 + F14*2 + F15*4 + F16*8 + F17*16 + F18*32 + F19*64 + F20*128
- ADD the ones you want **ON** together
- Add 1 for F13 ON
- Add 2 for F14 ON
- Add 4 for F15 ON
- Add 8 for F16 ON
- Add 16 for F17 ON
- Add 32 for F18 ON
- Add 64 for F19 ON
- Add 128 for F20 ON

- 0 Alone Turns OFF **F13-F20**
- ">" = End DCC++ EX command

To set functions F21-F28 on=(1) or off=(0): <f CAB BYTE1 [BYTE2]>

- "<" = Begin DCC++ EX command
- "f" = (lower case f) This command is for a CAB,s function.
- BYTE1: 223
- BYTE2: F21*1 + F22*2 + F23*4 + F24*8 + F25*16 + F26*32 + F27*64 + F28*128
- ADD the ones you want **ON** together
- Add 1 for F21 ON
- Add 2 for F22 ON
- Add 4 for F23 ON
- Add 8 for F24 ON
- Add 16 for F25 ON
- Add 32 for F26 ON
- Add 64 for F27 ON
- Add 128 for F28 ON
- 0 Alone Turns OFF **F21-F28**
- ">" = End DCC++ EX command

Returns: NONE

- CAB Functions do not have a Return
- CAB **Functions** do not get stored in the DCC++ EX CommandStation
- Each group does not effect the other groups. To turn on F0 and F22 you would need to send two separate commands to the DCC++ EX CommandStation. One for F0 on and another for F22 on.

STATIONARY ACCESSORY DECODERS & TURNOUTS

DCC++ EX COMMAND STATION can keep track of the direction of any turnout that is controlled by a DCC stationary accessory decoder once its Defined (Set Up).

All decoders that are not in a engine are accessory decoders including turnouts.

Besides being defined all turnouts, as well as any other DCC accessories connected in this fashion, can always be operated using the DCC COMMAND STATION Accessory command:

You Controlling a Accessory Decoder** with **< a ADDRESS SUBADDRESS ACTIVATE >

- "<" = Begin DCC++ EX command
- "a" (lower case a) this command is for a Acessory Decoder
- ADDRESS: the primary address of the decoder controlling this turnout (0-511)
- SUBADDRESS: the subaddress of the decoder controlling this turnout (0-3)
- ACTIVATE: (0) (Deactivate, Off, Unthrown) or (1) (Activate, On, Thrown)
- ">" = End DCC++ EX command
- However, this general command simply sends the appropriate DCC instruction packet to the main tracks to operate connected accessories. It does not store or retain any information regarding the current status of that accessory.

Defining (Setting up) a Turnout

To have the DCC++ EX CommandStation store and retain the direction of DCC-connected turnouts, as well as automatically invoke the required < a > command as needed, first define/edit/delete such turnouts using the following variations of the "T" command:

- Command to define a Turnout: **< T ID ADDRESS SUBADDRESS >**:
- Creates a new turnout ID, with specified ADDRESS and SUBADDRESS if turnout ID already exists, it is updated (over written) with the new specified ADDRESS and SUBADDRESS
- Returns: **< O >** if successful and **< X >** if unsuccessful (e.g. out of

memory)

- Command to Delete a turnout **< T ID >**:
- Deletes the definition of a turnout with this ID
- Returns: **< O >** if successful and **< X >** if unsuccessful (e.g. ID does not exist)
- Command to List all defined turnouts: **< T >**:
- Lists all defined turnouts.
- Returns: **< H ID ADDRESS SUBADDRESS THROW >** for each defined turnout or **< X >** if no turnouts have beed defined or saved.
- **ID**: The numeric ID (0-32767) of the turnout to control.
- (You pick the ID & They ares shared between Turnouts, Sensors and Outputs)
- **ADDRESS**: the primary address of the decoder controlling this turnout (0-511)
- **SUBADDRESS**: the subaddress of the decoder controlling this turnout (0-3)

Once all turnouts have been properly defined, Use the **< E >** command to store their definitions to EEPROM.
If you later make edits/additions/deletions to the turnout definitions, you must invoke the **< E >** command if you want those new definitions updated in the EEPROM.
You can also **ERASE everything (turnouts, sensors, and outputs)** stored in the EEPROM by invoking the **< e >** (lower case e) command.
(There is no Un-Delete)

> Example: You have a turnout on your main line going to warehouse industry. The turnout is controlled by a accessory decoder with a address of 123 and is wired to output 3. You want it to have the ID of 10.
> You would send the following command to the DCC++ EX CommandStation
> **< T 10 123 3 >**
>
> - This Command means:
> - <: Begin DCC++ EX command
> - T: (Upper case T) Define a Turnout

> - 10: ID number I am setting to use this turnout
> - 123: The accessory decoders address
> - 3: The turnout is wired to output 3
> - > : End DCC++ EX command
> > DCC-EX should return **< O >** Meaning Command Sucessful
> > Next you would send the following command to the DCC++ EX CommandStation
> > **< E >**
>
> - This Command means:
> - <: Begin DCC++ EX command
> - E: (Upper case E) Store (save) this definition to EEPROM
> - > : End DCC++ EX command
> > DCC++ EX should return **< O >** Meaning Command Successful

Controlling a Defined Turnout

- Sets turnout ID to either the "thrown"(turned) or "unthrown"(straight) position
- The Turnout format is **< T ID THROW >**
- ID: The numeric ID (0-32767) That you gave the turnout to control when you defined it.
- THROW: 0 (unthrown) or 1 (thrown)
- Returns: < H ID THROW > or < X > if turnout ID does not exist

> Example Continued from above:
> To throw turnout 10 so a engine can go to the warehouse siding you would send the following command.
> **< T 10 1 >**
>
> - This Command means:
> - <: Begin DCC++ EX command
> - T: (Upper case T) Throw a turnout.
> - 10: ID number of the defined turnout I want to control.
> - 1: Set turnout to Thrown (turned, on) position.

> - : End DCC++ EX command
> DCC++ EX should return **< H 10 1 >** Meaning Command Throw turnout 10 was Successful NOTE: This return may list all turnouts and thier directions

SENSORS

DCC++ EX CommandStation supports Sensor inputs that can be connected to any Aruidno Pin not in use by this program. Sensors can be of any type (infrared, magnetic, mechanical...). The only requirement is that when "activated" the Sensor must force the specified Arduino Pin LOW (i.e. to ground), and when not activated, this Pin should remain HIGH (i.e. 5V), or be allowed to float HIGH if use of the Arduino Pin's internal pull-up resistor is specified.

To ensure proper voltage levels, some part of the Sensor circuitry MUST be tied back to the same ground as used by the Arduino.

The Sensor code utilizes exponential smoothing to "de-bounce" spikes generated by mechanical switches and transistors. This avoids the need to create smoothing circuitry for each sensor. You may need to change the parameters in Sensor.cpp through trial and error for your specific sensors.

To have this sketch monitor one or more Arduino pins for sensor triggers, first define/edit/delete sensor definitions using the following variation of the "S" command:

- **< S ID PIN PULLUP >**: Creates a new sensor ID, with specified PIN and PULLUP if sensor ID already exists, it is updated with specified PIN and PULLUP (You choose the number).
- Returns: **< O >** if successful and **< X >** if unsuccessful (e.g. out of memory)
- **< S ID >**: Deletes definition of sensor ID
- Returns: **< O >** if successful and **< X >** if unsuccessful (e.g. ID does not exist)
- **< S >**: Lists all defined sensors
- Returns: **< Q ID PIN PULLUP >** for each defined sensor or if no sensors defined

ID: The numeric ID (0-32767) of the sensor
(You pick the ID & They ares shared between Turnouts, Sensors and Outputs)
PIN: The Arduino pin number the sensor is connected to on the Arduino board.
PULLUP: 1 = Use internal pull-up resistor for PIN, 0 = don't use internal pull-up resistor for PIN

Once all sensors have been properly defined, use the **< E >** (upper case E) command to store their definitions to EEPROM.
If you later make edits/additions/deletions to the sensor definitions, you must invoke the **< E >** (upper case E) command if you want those new definitions updated in the EEPROM.
You can also clear **everything (turnouts, sensors, and outputs)** stored in the EEPROM by invoking the **< e >** (lower case e) command.
(There is NO UN-Delete)

All sensors defined as per above are repeatedly and sequentially checked within the main loop of this sketch.If a Sensor Pin is found to have transitioned from one state to another, one of the following serial messages are generated:

- **< Q ID >** - for transition of Sensor ID from HIGH state to LOW state (i.e. the sensor is triggered)
- **< q ID >** - for transition of Sensor ID from LOW state to HIGH state (i.e. the sensor is no longer triggered)

Depending on whether the physical sensor is acting as an "event-trigger" or a "detection-sensor," you may decide to ignore the **< q ID >** return and only react to **< Q ID >** triggers.

ARDUINO OUTPUT PINS

DCC++ EX CommandStation supports optional OUTPUT control of any unused Arduino Pins for custom purposes. Pins can be activited or de-activated. The default is to set ACTIVE pins HIGH and INACTIVE pins LOW. However, this default behavior can be inverted for any pin in which case ACTIVE=LOW and INACTIVE=HIGH.

Definitions and state (ACTIVE/INACTIVE) for pins are retained in EEPROM and restored on power-up.
The default is to set each defined pin to active or inactive according to its restored state. However, the default behavior can be modified so that any pin can be forced to be either active or inactive upon power-up regardless of its previous state before power-down.

To have DCC++ EX CommandStation utilize one or more Arduino pins as custom outputs, first define/edit/delete output definitions using the following variation of the "Z" command:

- **< Z ID PIN IFLAG >**: Creates a new output ID, with specified PIN and IFLAG values.
  - if output ID already exists, it is updated with specificed PIN and IFLAG.
  - Note: output state will be immediately set to ACTIVE/INACTIVE and pin will be set to HIGH/LOW according to IFLAG value specifcied (see below).
  - Returns: **< O >** if successful and **< X >** if unsuccessful (e.g. out of memory).
- **< Z ID >**: Deletes definition of output ID
- Returns: **< O >** if successful and **< X >** if unsuccessful (e.g. ID does not exist)
- **< Z >**: Lists all defined output pins
  - Returns: **< Y ID PIN IFLAG STATE >** for each defined output pin or **< X >** if no output pins defined.

ID: The numeric ID (0-32767) of the output
(You pick the ID & They ares shared between Turnouts, Sensors and Outputs)
PIN: The Arduino pin number to use for the output.
STATE: The state of the output (0=INACTIVE / 1=ACTIVE)
IFLAG: Defines the operational behavior of the output based on bits 0, 1, and 2 as follows:

```
IFLAG, bit 0: 0 = forward operation (ACTIVE=HIGH /
INACTIVE=LOW)
```

```
              1 = inverted operation (ACTIVE=LOW /
INACTIVE=HIGH)

IFLAG, bit 1: 0 = state of pin restored on power-up to
either ACTIVE or INACTIVE
                  depending on state before power-down.
              1 = state of pin set on power-up, or when
first created,
                  to either ACTIVE of INACTIVE depending on
IFLAG, bit 2

IFLAG, bit 2: 0 = state of pin set to INACTIVE uponm power-
up or when first created
              1 = state of pin set to ACTIVE uponm power-up
or when first created
```

Once all outputs have been properly defined, use the **< E > Upper Case E** command to store their definitions to EEPROM.
If you later make edits/additions/deletions to the output definitions, you must invoke the **< E >** command if you want those new definitions updated in the EEPROM.
You can also **ERASE everything (turnouts, sensors, and outputs)** stored in the EEPROM by invoking the **< e >** (lower case e) command.
(There is no Un-Delete)

To change the state of outputs that have been defined use:

- **< Z ID STATE >**: Sets output ID to either ACTIVE or INACTIVE state
- Returns: **< Y ID STATE >**, or **< X >** if output ID does not exist
  - **ID**: The numeric ID (0-32767) of the defined output to control
  - **STATE**: The state of the output (0=INACTIVE / 1=ACTIVE)

When controlled as such, the Arduino updates and stores the direction of each output in EEPROM so that it is retained even without power. A list of the current states of each output in the form **< Y ID STATE >** is generated by DCC++ EX CommandStation whenever the **< s >** status command is invoked. This provides an efficient way of initializing the state of any outputs being monitored or controlled by a separate interface or GUI program.

Engine Decoder Programming Commands

PROGRAMMING-MAIN TRACK

WRITE CV BYTE TO ENGINE DECODER ON MAIN TRACK

Writes, without any verification, a Configuration Variable BYTE to the decoder of an engine on the main operations track.

- Write CV BYTE Format is: **< w CAB CV VALUE >**
- **CAB**: The short (1-127) or long (128-10293) address of the engine decoder
- **CV**: The number of the Configuration Variable memory location in the decoder to write to (1-1024)
- **VALUE**: The value to be written to the Configuration Variable memory location (0-255)
- Returns: NONE

WRITE CV BIT TO ENGINE DECODER ON MAIN TRACK

Writes, without any verification, a single bit within a Configuration Variable BIT to the decoder of an engine on the main operations track.

- Write CV BIT Format is: **< b CAB CV BIT VALUE >**
- **CAB**: the short (1-127) or long (128-10293) address of the engine decoder
- **CV**: the number of the Configuration Variable memory location in the decoder to write to (1-1024)
- **BIT**: the bit number of the Configurarion Variable regsiter to write (0-7)
- **VALUE**: the value of the bit to be written (0-1)
- Returns: NONE

PROGRAMMING-PROGRAMMING TRACK

WRITE CV BYTE TO ENGINE DECODER ON PROGRAMMING TRACK

Writes, and then verifies, a Configuration Variable BYTE to the decoder of an engine on the programming track

- Write CV BYTE Format is: **< W CV VALUE CALLBACKNUM CALLBACKSUB >**
- **CV**: The number of the Configuration Variable memory location in the decoder to write to (1-1024).
- **VALUE**: The value to be written to the Configuration Variable memory location (0-255).
- **CALLBACKNUM**: An arbitrary integer (0-32767) that is ignored by the Command Station and is simply echoed back in the output - useful for external programs that call this function.
- **CALLBACKSUB**: a second arbitrary integer (0-32767) that is ignored by the Command Station and is simply echoed back in the output - useful for external programs (e.g. DCC++ EX Interface) that call this function.

- Returns: **< r CALLBACKNUM|CALLBACKSUB|CV Value >**
- CV VALUE: Is a number from 0-255 as read from the requested CV, or -1 if verification read fails.

WRITE CV BIT TO ENGINE DECODER ON PROGRAMMING TRACK

Writes, and then verifies, a Configuration Variable BIT to the decoder of an engine on the programming track

- Write CV BIT Format is: **< B CV BIT VALUE CALLBACKNUM CALLBACKSUB >**
- **CV**: The number of the Configuration Variable memory location in the decoder to write to (1-1024).
- **BIT**: The bit number of the Configuration Variable memory location to write (0-7).
- **VALUE**: The value of the bit to be written (0-1).
- **CALLBACKNUM**: An arbitrary integer (0-32767) that is ignored by the Command Station and is simply echoed back in the output - useful for external programs that call this function.
- **CALLBACKSUB**: A second arbitrary integer (0-32767) that is ignored by the Command Station and is simply echoed back in the output -

useful for external programs (e.g. DCC++ EX Interface) that call this function.

- Returns: **< r CALLBACKNUM|CALLBACKSUB|CV BIT VALUE>**
- CV VALUE is a number from 0-1 as read from the requested CV bit, or -1 if verification read fails.

READ CONFIGURATION VARIABLE BYTE FROM ENGINE DECODER ON PROGRAMMING TRACK

Reads a Configuration Variable from the decoder of an engine on the programming track.

- Read CV BYTE Format is:**< R CV CALLBACKNUM CALLBACKSUB >**
- **CV**: The number of the Configuration Variable memory location in the decoder to read from (1-1024).
- **CALLBACKNUM**: An arbitrary integer (0-32767) that is ignored by the Command Station and is simply echoed back in the output - useful for external programs that call this function.
- **CALLBACKSUB**: A second arbitrary integer (0-32767) that is ignored by the Command Station and is simply echoed back in the output - useful for external programs (e.g. DCC++ EX Interface) that call this function.

- Returns: **< r CALLBACKNUM|CALLBACKSUB|CV VALUE>
- CV VALUE is a number from 0-255 as read from the requested CV, or -1 if read could not be verified.

SEND PACKET TO THE TRACK

Command - Writes a DCC packet of two, three, four, or five hexidecimal bytes to a register driving the main operations track

FORMAT: <M REGISTER BYTE1 BYTE2 [BYTE3] [BYTE4] [BYTE5]>

NOTE: THIS IS FOR DEBUGGING AND TESTING PURPOSES ONLY. DO NOT USE UNLESS YOU KNOW HOW TO CONSTRUCT NMRA DCC PACKETS - YOU CAN INADVERTENTLY RE-PROGRAM YOUR ENGINE DECODER

```
REGISTER: an internal register number, from 0 through
MAX_MAIN_REGISTERS (inclusive), to write (if REGISTER=0) or
write and store (if REGISTER>0) the packet
BYTE1:  first hexadecimal byte in the packet
BYTE2:  second hexadecimal byte in the packet
BYTE3:  optional third hexadecimal byte in the packet
BYTE4:  optional fourth hexadecimal byte in the packet
BYTE5:  optional fifth hexadecimal byte in the packet

returns: NONE
```

For more information please visit our webpage at DCC++ EX Webpage