MERG Wiki

The latest published version can be downloaded. merg_cbus_modules_1d.pdf NB this link to be removed when the wiki becomes the definitive document.

MERG CBUS Modules.

Changes from Rev 1d 01/02/2014 Imported to Wiki from MS word, reformatted and some re-arrangement.

Changes from Rev 1c 10/09/2013 Update to CAB operation for firmware 2r and later.

A description and developer's guide to the MERG CBUS modules.

Description of the current MERG CBUS modules.

This document should be used in conjunction with the main Developer's Guide

These give the information needed for software development and do not include a hardware description in any detail.

1. Layout modules

```
1.1     CANACC4 and CANACC4_2           4 output solenoid driver
1.2     CANACC5 and CANACC8             8 output drivers
1.3     CANACEBC/CANTOTI                8 input module
1.4     CANACE3                         128 switch scanner
1.5     CANLED and CANLED64             64 LED driver
1.6     CANSERV0                        8 output servo driver
1.7     CANSERV08C                      8 output servo driver with position
                                              feedback events
```

The DCC CABs and Command Station details are included in Section 2.(Link from list on right).

Unless stated otherwise the following descriptions relate to the current released versions as listed in the module pages listed here.

1.1 CANACC4 and CANACC4_2

This module drives 4 solenoid type point motors and incorporates a Capacitor Discharge Unit (CDU). The two models are identical in configuration and operation. The only difference is in the power supply requirements. The CANACC4 uses 15V AC and the CANACC4_2 uses a 12V DC supply. Note that the CANACC4 15V AC hardware can run the CANACC4_2 firmware. It will then identify itself as a CANACC4_2.

Node variables

```
9   (NV# of 1 to 9)
```

Each NV sets the pulse duration of its respective output. The increments are in 10millisec. intervals. A value of 0 gives a continuous ON output and should not be used for pulse outputs. Even though this module only defines 4 output pairs, it is possible to have different times for each side of the pair. The default NV setting is 5 for all outputs (50 millisec pulses). NV 9 is used for setting the recharge timer in CANACC4_2. This is the time the firmware waits before firing successive outputs. It allows time for the CDU to recharge, the default is 20 (200ms). This NV is only used from version J of the firmware, setting it for earlier versions will have no effect.

Number of stored events 32 or 128 depending on the firmware version (EN# of 1 to 32 or 128)

Number of EVs per event 2. (EV# of 1 or 2)

EV1 determines which output pair the event applies to. Allowed values of 00000001 to 00001111 where each bit corresponds to one of the four outputs. A bit set makes that output pair active. If an event fires more than one solenoid, the module incorporates its own delay to allow the CDU to recharge and the outputs fire sequentially.

EV2 determines the polarity of each active output pair. Allowed values of 00000000 to 00001111. A bit set reverses the output pair relative to the ON or OFF OpCode.

Supported OpCodes (HEX value and mnemonic) in FLiM mode.

HEX	Mnemonic		HEX	Mnemonic		HEX	Mnemonic
*0D	QNN		59	WRACK		96	NVSET
10	RQNP		5C	BOOTM		97	NVANS
*11	RQMN		*5D	ENUM		98	ASON
42	SNN		6F	CMDERR		99	ASOF
50	RQNN		70	EVNLF		9B	PARAN
51	NNREL		71	NVRD		9C	REVAL
52	NNACK		72	NENRD		B2	REQEV
53	NNLRN		73	RQNPN		B5	NEVAL
54	NNULN		74	NUMEV		*B6	PNN
55	NNCLR		*75	CANID		D2	EVLRN
56	NNEVN		90	ACON		D3	EVANS
57	NERD		91	ACOF		*E2	NAME
58	RQEVN		95	EVULN		EF	PARAMS
						F2	ENRSP

* These OpCodes are introduced from Major Version 2.

The module ID number is 1 and the manufacturer number is 165.

Note: Even in SLiM mode, it is possible to read the node parameters over the bus. A SLiM node has a default NN of 00 00 so a RQNPN command to NN of 00 00 will return the indexed parameter from that node. This may be useful for reading the code revision. Only one SLiM mode node can be active for this process or you will get a response from all of them. The CAN_ID will also be 00 00.

For bootloading in SLiM mode, the NN of 00 00 must be used and, again, only one SLiM mode module can be active for this.

1.2 CANACC5 or CANACC8

These modules drive 8 separate outputs. The CANACC5 has high current bipolar output drivers (source or sink), nominally 1 amp each but limited by the power supply and IC dissipation. Recommended current drain is 1 amp total for continuous operation.

The CANACC8 has open collector outputs so are current sink only. These are rated at 0.5 amps per output but again limited by the power supply and IC dissipation to a recommended 1 amp total.

The two models are identical in configuration and operation.

There are now two variants of the firmware available for this module. The basic version is Rev u. This will be described first.

Node variables 0

The basic version has no node variables. All outputs are either ON or OFF.

Number of stored events 32 or 128 depending on the firmware version (EN# of 1 to 32 or 128)

Number of EVs per event 2. (EV# of 1 or 2)

EV1 determines which output the event applies to. Allowed values of 00000000 to 11111111 where each bit corresponds to one of the 8 outputs. A bit set makes that output active.

EV2 determines the polarity of each active output. Allowed values of 00000000 to 11111111. A bit set reverses the output relative to the ON or OFF OpCode. For operation in pairs as would be required for motorised turnout drivers, set both of the pair outputs ON but use different values in EV2, e.g. for outputs 1 and 2 as a pair, EV1 = 00000011 and EV2 = 00000001 or EV2 = 00000010 depending on the direction required.

Supported OpCodes (HEX value and mnemonic) in FLiM mode.

HEX	Mnemonic		HEX	Mnemonic		HEX	Mnemonic
			59	WRACK			
10	RQNP		5C	BOOTM			
						98	ASON
42	SNN		6F	CMDERR		99	ASOF
50	RQNN		70	EVNLF		9B	PARAN
51	NNREL					9C	REVAL
52	NNACK		72	NENRD		B2	REQEV
53	NNLRN		73	RQNPN		B5	NEVAL
54	NNULN		74	NUMEV			
55	NNCLR					D2	EVLRN
56	NNEVN		90	ACON		D3	EVANS
57	NERD		91	ACOF			
58	RQEVN		95	EVULN		EF	PARAMS
						F2	ENRSP

Note: Even in SLiM mode, it is possible to read the node parameters over the bus. A SLiM node has a default NN of 00 00 so a RQNPN command to NN of 00 00 will return the indexed parameter from that node. This may be useful for reading the code revision. Only one SLiM mode node can be active for this process or you will get a response from all of them. The CAN_ID will also be 00 00.

For bootloading in SLIM mode, the NN of 00 00 must be used and, again, only one SLiM mode module can be active for this.

New version of the CANACC5 and CANACC8 code (From Rev V) This code has a set of Node Variables which allow individual outputs to be pulsed or flashing. There are 8 NVs, one for each output.

Node variables 8 (NV# of 1 to 8)

The Node Variable values can range from 0 to 127, 0 means a continuous output, a none zero value is the pulse width in units of 20ms, so a value of 25 will give a 1/2 second single pulse. If you add 128 (set bit 7) to the value then the pulse is repeated continuously, so a value of 153 will give repeated 1/2 second pulses.
Single pulses can be generated by either On or Off events, but repeating pulses can only be started by an On event, an Off event stops the pulses.
If an event is configured to drive more than one output and the outputs are set to have repeating pulses, then outputs selected for On will repeat in anti-phase to outputs selected for Off, e.g. for crossing lights.

With NV values set to 0 (the default), Rev V acts like Rev U. The ability to set different values for the NVs is only available in FLiM mode.

The relative OpCodes for NV manipulation are now included in this version.

Supported OpCodes (HEX value and mnemonic) in FLiM mode.

HEX	Mnemonic		HEX	Mnemonic		HEX	Mnemonic
*0D	QNN		59	WRACK		96	NVSET
10	RQNP		5C	BOOTM		97	NVANS
*11	RQMN		*5D	ENUM		98	ASON
42	SNN		6F	CMDERR		99	ASOF
50	RQNN		70	EVNLF		9B	PARAN
51	NNREL		71	NVRD		9C	REVAL
52	NNACK		72	NENRD		B2	REQEV
53	NNLRN		73	RQNPN		B5	NEVAL
54	NNULN		74	NUMEV		*B6	PNN
55	NNCLR		*75	CANID		D2	EVLRN
56	NNEVN		90	ACON		D3	EVANS
57	NERD		91	ACOF		*E2	NAME
58	RQEVN		95	EVULN		EF	PARAMS
						F2	ENRSP

1.3 CANACE8C/CANTOTI

Hardware

The CANACE8C/CANTOTI modules run on the CANACE8C board, available as a MERG kit, which requires a power supply of 5V DC and also on the CANACE8C_2 board which will take a 12V DC supply.

Overview

These modules have 8 inputs. The inputs have resistive pullups to 5V so are normally logic high. If switched to their 0V line, they give a logic low. They will also work with 5V logic inputs and have input protection against higher voltages. The pullup value is 100K so they will work well with opto-isolators or other similar open collector devices.

A logic high is considered an OFF state and a logic low an ON state for CBUS purposes.

In addition to producing a CBUS event when an input changes, these modules can have their inputs polled by another module or PC so their input states can be determined at any time. They can be taught an event which, when received, will prompt a sequence of 8 events reflecting the input states. This is referred to as the Start of Day (SoD) event. It would be used to determine the complete layout state on power up or at any other time required. An additional mode (mode 1) of learned events prompts a single response event (long) where the LSbyte reflects the 8 inputs. To distinguish this from a long event caused by an input change, bit 0 of byte 1 is also set. i.e. an EN of 01 00 to 01 FF. Note that a high input produces a '1' bit, a low input produces a '0' bit. A possible use of this is for route setting with rotary switches attached to the inputs. The LSbyte is the number set on the switches.

The default state of a CANACE8C/CANTOTI is the sending of long events when an input is changed. The upper two bytes of the event are the Node Number and the lower two bytes, the input number. The inputs are numbered 1 to 8. The events are the usual ON or OFF (0x90 or 0x91). However, individual inputs can be allocated Device Numbers (DNs) for the short event (addressed) mode. This requires teaching the module the DN corresponding with each input. If an input has been given a DN, it will only produce short events with that DN when the input changes or on a SoD event. The DN has a two byte range.

Node variables: 1 (NV# = 1, 6 or 7 depending on firmware version)

NV1, NV2 and NV3 are byte values with bits 0 to 7 corresponding to inputs 1 to 8. The default values are NV1 = 0, NV2 = 0, NV3= 0

NV1 is a byte where each bit determines whether a particular input sends ON / OFF or ON only events. Inputs 1 to 8 correspond to NV1 bits 0 to 7. A 1 bit indicates an ON only event. The default is ON / OFF.

NV2 is a byte value that determines whether an input is to be inverted giving an active-high input.. The default is low-active.

To allow for noisy signals being connected to the CANACE8C/CANTOTI, optional delays can be enabled.

NV3 is a byte value which enables the delay on each of specified inputs

If input delay is enabled for an input, the input must be continuously ON for at least the ON delay time before an ON event is generated, and must be continuously OFF for at least the OFF delay time before an OFF event is generated.

NV4, the ON delay time in units of 10ms, the default value is 10 (100ms).

NV5, the OFF delay in units of 10ms, the default value for a CANACE8C is 10 (100ms) and for a CANTOTI is 50 (500ms)

For modules that support 6 NVs, NV6 is not used, reserved for future expansion.

For modules that support 7 NVs, NV6 sets inputs to Push Button Toggle mode, inputs 1 to 8 correspond to NV6 bits 0 to 7. When a bit is set, the corresponding input will alternate between sending ON and OFF events each time the input state changes from high to low. NV7 is not used and reserved for future use.

Number of stored events: 32 (EN# of 1 to 32)

Number of EVs per event: 2. (EV# of 1 or 2)

EV1 sets the mode for the learned event. EV1 = 0 is for the SoD event. EV1 = 1 is for the 'route' event. EV1 of 00001000 to 00001111 relate the event to a particular input. This is used when teaching a DN for a given input and also for polling a particular input by its DN.

EV2 The code allows for an additional EV but this is not currently used.

Supported OpCodes

This table shows the supported OpCodes (HEX value and mnemonic) in FLiM mode.

HEX	Mnemonic		HEX	Mnemonic		HEX	Mnemonic
*0D	QNN		59	WRACK		96	NVSET
10	RQNP		5C	BOOTM		97	NVANS
*11	RQMN		*5D	ENUM		98	ASON
42	SNN		6F	CMDERR		99	ASOF
50	RQNN		70	EVNLF		9A	ASRQ
51	NNREL		71	NVRD		9B	PARAN
52	NNACK		72	NENRD		9C	REVAL
53	NNLRN		73	RQNPN		9D	ARSON
54	NNULN		74	NUMEV		9E	ARSOF
55	NNCLR		*75	CANID		B2	REQEV
56	NNEVN		90	ACON		B5	NEVAL
57	NERD		91	ACOF		*B6	PNN
58	RQEVN		92	AREQ		D2	EVLRN
			93	ARON		D3	EVANS
			94	AROF		*E2	NAME
			95	EVULN		EF	PARAMS
						F2	ENRSP

* These OpCodes are introduced from firmware Major Version 2

Module ID The module ID number is 5 for the CANACE8C and 17 for the CANTOTI. The manufacturer number is 165.

Notes Even in SLiM mode, it is possible to read the node parameters over the bus. The NN must be the value set by the switches and links. This may be useful for reading the code revision. In SliM mode the CAN_ID will also be . the same as the NN

For bootloading in SLIM mode, the NN as set by the switches and links must be used.

1.4 CANACE3 and CANACE3_2

CANACE3 is intended for use with control panels and can scan 128 switches via a matrix arrangement. It is a producer module but you can teach any switch input to send any event, including short events, hence you can have the switches as numbered 'devices' or have more than one control panel sending the same events.

CANACE3 is a 15V AC powered module and CANACE3_2 uses a 12V DC supply. The firmware is identical for both.

The switch matrix scanning method.

The control panel switches are interrogated using a conventional diode matrix method. This allows a large number of switches to be connected with relatively little wiring. Only 24 wires for 128 switches or 64 PB pairs. The matrix is arranged as 16 columns and 8 rows. Each column is pulsed from 5V (high) to 0V (low) in turn on a cyclical basis. The rows are connected to the columns via a switch and a series diode. If a switch is closed, the corresponding row is taken low when the column goes low. The voltage drop across the series diode still gives a logic low with the CMOS inputs of the PIC processor. Unlike similar arrangements for keypad scanning where only one button is pressed at one time, the diodes are required as switches may remain closed. If only pushbuttons are used, the diodes may be omitted.

The processor firmware detects changes in the switches or PBs and sends a corresponding CBUS event. When in switch mode, a switch closure sends an ON event and a switch opening sends an OFF event. In accord with the CBUS protocol, the event is a 32 bit (4 byte) number. The upper two bytes are the Node Number, in the case of CANACE3 in SLiM mode, this range is 1 to 4 only. The lower two bytes are the node event (switch number). Again, for CANACE3, these numbers start at 1 and run sequentially to 128 (or 1 to 64 for the dual pushbutton mode). Event or NN of zero is not used. With the present PIC code, it is now possible to allocate different events to individual switches which supplant the default values. If these are taught as 'short' events, you can give each switch a 'device number' which may be useful for PC operation. This alternative allocation requires FLiM mode and a suitable configuration tool over the CBUS network.

To further simplify the control panel wiring, the numbering sequence starts at 1 with column 1 and row 1, 2 with column 1 and row 2, 3 with column 1 and row 3 and 4 with column 1 and row 4. However, event 5 is column 2 and row 1 (not column 1 and row 5) so only the first 4 rows are used for events 1 to 64 and the second set of rows (5 to 8) for events 65 to 128. This allows for a sequential set of events for smaller control panels using only 4 row wires total. The same applies in PB mode but each column generates 2 events for the 4 rows so rows 1 to 4 give events 1 to 32 and rows 5 to 8 give events 33 to 64. All matrix wiring is conveniently supplied via a 25 way D type socket. The pinout is given on the schematic.

Selection of switch mode or push button mode is via the jumper J5. J5 has three pins. Placing the jumper between the centre and lower pins sets 'switch' mode and between the centre and upper pins, selects the PB mode. (Viewed with the regulator U2 at the top) When in FLiM mode, this jumper is ignored.

Configuring in SLiM mode. The CANACE3 module has two jumpers (or DIL switch if fitted instead) to select its Node Number. Due to the pinout limitation of the PIC processor, only 4 NNs are possible. The two jumpers select 0 to 3 although this is converted to NNs of 1 to 4 by the firmware. This limits the number of CANACE3 modules in SLiM mode to 4 on a layout. As with other SLiM modules, the NN also becomes the CAN_ID. By using the jumper J5, you can select between 128 ON / OFF toggle switches or 64 pushbutton pairs.

Teaching a switch event or number.

The default events sent by a CANACE3 are 'long' events with the module NN as the upper two bytes and the switch number as the lower two. However, if you want to send short events or different long events, you can teach these in SLiM. A brief press of the on-board small pushbutton will cause the green LED to flash. Activate the switch or PB you wish to change. The direction or which of the PB pair doesn't matter. Send an ON or OFF event you wish that switch to have. The green LED will now go steady again and the switch will send the taught event. If the taught event comes from another control panel, you can have the same event created by both control panels.

Note: The Node parameters can be read in SLiM mode using RQNPN (0x73) with the NN as set by the links

Configuring in FLiM mode.

FLiM mode allows for greater possibilities of switch or pushbutton combinations than in SLiM. This is done with a Node Variable.

Node variables 1 (NV# = 1)

NV1 is a byte which sets the possible modes for the switch or PB combinations.

Mode 0 All switches work as ON / OFF toggles (same as SLiM with the jumper to 0v)

Mode 1 All pushbutton pairs (same as SLiM with the jumper to +5V)

Mode 2 Top 4 rows Toggles, bottom 4 rows PB pairs

Mode 3 Top 4 rows Toggles, bottom 4 rows PB ON only

Mode 4 Top 4 rows PB pairs, bottom 4 rows PB ON only

Mode 5 All PB ON only

The NV is a value corresponding to the above Mode.

Teaching the switch events.

The CANACE3 has no stored events as it is a producer only module. However, you can teach each switch or PB to have its own event which it sends when activated. This overrides the default value. Each switch has its own index value (switch number) defined by its position in the matrix. However, as it is not easy to know which switch is connected to which row / column, the teaching of alternative switch 'events' has been simplified in a similar way to that used in SLiM mode. When a CANACE3 is put into learn mode (<0x53><NNhi><NNlo>), you can activate a switch or PB on the panel. This will send the current switch event but with an added byte. This added byte is the switch number or index value. As the OPC for this is different, actual layout devices should not respond as if it were just a switch change. If it is a ON event, you get

<0xB0><NNhi><NNlo><ENhi><ENlo><EN#>

Where EN# is now the switch number. This can be used to teach the switch to send a different event, including a short event. In this latter case it amounts to giving a switch a Device Number (DN). If using a PC to control the layout, this DN should be a unique value for that switch. The PC will now recognise a switch change by its DN. If the switch is to control a device directly, the DN you give it should be that of the device to be controlled. This is the reverse process to teaching a consumer to recognise a long event produced by the control panel. You can also teach a switch to send a different 'long' event in the situation where you want two or more control panels to send the same event as might be the case for a SoD event. You teach the switch event in the same way as for teaching any other event in indexed mode except there is no Event Variable (EV) associated with it.

<0xF5><NNhi><NNlo><ENhi><ENlo><EN#><00><00>

EN# is the switch number. For a short event (DN), use

<0xF5><00 ><00><DNhi><DNlo><EN#><00><00>

As this is a stored event, the CANACE3 will respond with a WRACK and then should be taken out of learn mode. All switch events can be reset to the default value by a NNCLR.

It is not possible (at present) to poll the switches for their settings. You can read back a switch event number using NENRD provided the switch number is known.

Supported OpCodes

HEX	Mnemonic		HEX	Mnemonic		HEX	Mnemonic
*0D	QNN		59	WRACK		96	NVSET
10	RQNP		5C	BOOTM		97	NVANS
*11	RQMN		*5D	ENUM		98	ASON
42	SNN		6F	CMDERR		99	ASOF
50	RQNN		70	EVNLF		9B	PARAN
51	NNREL		71	NVRD		9C	REVAL
52	NNACK		72	NENRD		B0	ACON1
53	NNLRN		73	RQNPN		B2	REQEV
54	NNULN		74	NUMEV		B5	REVAL
55	NNCLR		*75	CANID		*B6	PNN
			90	ACON		*E2	NAME
			91	ACOF		EF	PARAMS
						F2	ENRSP
						F5	EVLRNI

* These OpCodes are introduced from firmware Major Version 2

The module ID number is 4 for the CANACE3. The manufacturer number is 165.

Note: Even in SLiM mode, it is possible to read the node parameters over the bus. A RQNPN command to the NN as set on the two jumpers will return the indexed parameter from that node. This may be useful for reading the code revision.

For bootloading in SLIM mode, the NN as set by the switches must be used.

1.5 CANLED

The CANLED module is intended for driving up to 64 LEDs on a control panel. Although intended for control panel use, it does not preclude its use for any other LED application such as colour light signals, traffic lights or building lighting.

The hardware design uses a matrix arrangement for driving the LEDs. There are 4 row lines and 16 column lines. The column lines are constant current sinks so no series resistors are needed for the LEDs. The current, and hence brightness can be programmed with a resistor although individual LEDs all have the same current. The LED anodes must be connected to the row lines and the anodes to the column lines.

While the hardware is common, there are two variants of the firmware. The original CANLED2 code only allows one LED to be controlled (on or off) by a single event. In SLiM mode, the LED is selected by a 6 way DIL switch, giving one of 64. The polarity can be changed and also there is a facility for operating successive LEDs in pairs, as might be used for turnout position indication.

CANLED64 is a more recent variant that allows an event to set/clear any combination of LEDs. As this now needs 64 options per event, it is only practicable to use it in FLiM mode. The 'paired' mode is not used as each LED can be set on or off for every event. The two variants will be described separately.

CANLED2

Node variables 0

The basic version has no node variables. All LEDs are either ON or OFF.

Number of stored events 248 (EN# of 1 to 248)

Number of EVs per event 1 (EV# of 1)

EV1 determines which LED the event applies to and also the polarity and toggle mode. The lower 6 bits (0 to 5) set which LED the event applies to. One binary number per LED. Bit 6, if set reverses the polarity and bit 7, if set toggles between the LED number and the next one up.

As the CANLED is a consumer only, it has a default SLiM NN of 00 00. This is used for bootloading and readback of the node parameters.

Setting up in SLiM mode.

The module has a 6 way DIL switch for setting the LED the event is to apply to and four jumpers in a block labelled Ln,Un,Tg and Po. (Learn, Unlearn, Toggle and Polarity). The DIL switch sets binary numbers from 0 to 63 although for convention, the LEDs are numbered from 1 to 64. Select the LED required on these switches. Put the Ln jumper in. Send the event to be learned. This can be a long or short event. If you want that event to turn on the numbered LED and turn off the next one, put the Tg jumper in. To reverse the LED, put the Po jumper in as well. Then remove the Ln jumper. To unlearn that event, put both LN and Un jumpers in and send the event. To clear all events, put just the Un jumper in and cycle the power.

Setting up in FLiM mode.

FLiM mode setting is very similar to SLiM mode. Assuming the module has a NN and is already in FLiM mode, send the NNLRN to put the module into learn mode. Send the event(s) to be taught using EVLRN. This can be a long or short event. If a short event, it gives a LED a 'device number'. There is one EV attached to each event. (EV# = 1). The lower 6 bits (0 to 5) of the EV make up the LED number, in the same way as for the DIL switches in SLiM mode. The top two bits correspond to the toggle and polarity jumpers. Bit 6 is toggle mode and bit 7 is polarity. Then take out of learn mode with NNULN. All events can be cleared with NNCLR.

Supported OpCodes (HEX value and mnemonic) in FLiM mode.

HEX	Mnemonic		HEX	Mnemonic		HEX	Mnemonic
*0D	QNN		59	WRACK			
10	RQNP		5C	BOOTM			
*11	RQMN		*5D	ENUM		98	ASON
42	SNN		6F	CMDERR		99	ASOF
50	RQNN		70	EVNLF		9B	PARAN
51	NNREL					9C	REVAL
52	NNACK		72	NENRD		B2	REQEV
53	NNLRN		73	RQNPN		B5	NEVAL
54	NNULN		74	NUMEV		*B6	PNN
55	NNCLR		*75	CANID		D2	EVLRN
56	NNEVN		90	ACON		D3	EVANS
57	NERD		91	ACOF		*E2	NAME
58	RQEVN		95	EVULN		EF	PARAMS
						F2	ENRSP

The module ID number is 6 for the CANLED2. The manufacturer number is 165.

Note: Even in SLiM mode, it is possible to read the node parameters over the bus. A SLiM node has a default NN of 00 00 so a RQNPN command to NN of 00 00 will return the indexed parameter from that node. This may be useful for reading the code revision. Only one SLiM mode node can be active for this process or you will get a response from all of them. The CAN_ID will also be 00 00.

For bootloading in SLIM mode, the NN of 00 00 must be used and, again, only one SLiM mode module can be active for this.

CANLED64

This is a new code for the CANLED module. The upgrade arose from the desire to have many LEDs set or cleared by a single event, as would occur when setting routes. The problem was that each event now needed 64 bits for each event to identify the LEDs associated with that event and another 64 bits for each LED polarity. As individual LEDs could now be set ON or OFF with one event, the Toggle facility was not needed. While it is possible to set up a CANLED64 in SLiM mode, it is so tedious that only the FLiM setup will be described.

Events are stored in Flash Memory, as does the CANLED2, but the data structure is different. The number of stored events now is 255 (EN# of 1 to 255) and the EVs per event is now 17.

Node variables 0

Number of stored events 255 (EN# of 1 to 255)

Number of EVs per event 17 (EV# of 1 to 17)

The EVs from 1 to 8 are the 8 bytes representing the bit pattern of the 64 LEDs for that event, one bit per LED. EVs 9 to 16 are the corresponding polarity bits for each LED. A 0 is normal polarity, a 1 is reversed. EV# 17 defines the 'effects' associated with the event such as LEDs flashing.

EV# 17 has the following values

0xFF (255) Normal operation, both On and Off events behave normally. 0xFE (254) only On Events are actioned. 0xFD (253) only Off Events are actioned. 0xF8 (248) the event causes the selected LEDs to flash, LEDs set with a corresponding polarity bit will flash in anti-phase to normal LEDs. An On event starts the flashing, and Off event stops the flashing.

The last setting enables pairs of LEDs to flash alternately with the same event.

As only one EV can be taught per EVLRN message, it requires the configuration software to send the event 17 times, incrementing the EV# each time.

If taught as a short event, the 'device number' corresponds to the LED pattern rather than to individual LEDs although if only one LED is selected out of the possible 64, it can result in a DN for a LED. Given the possible 255 events, you can define a DN for each of the LEDs (64 short events) and have other DNs defining complete routes or lighting combinations etc.

The same OpCodes are used as for the CANLED2 version.

The module ID number is 7 for the CANLED64.

1.6 CANSERVO and CANSERVO8C

The original CANSERVO has been enhanced with additional functions to give the CANSERVO8C. The CANSERVO is described first followed by the CANSERVO8C.
CANSERVO
The CANSERVO is an alternative code that will run on a CANACC8 module or on the CANSERVO module. When used with the CANACC8 module, the output IC (U4), which is a Darlington driver, should be replaced with a resistor array (resnet) of 8 x 1K resistors. The 12V version of the CANACC8 (ACC8_2) has output options for either the Darlington array or the resnet. On this board, the jumpers for the outputs should be set to the resistor network. The same code can be used in the CANACC5 board but this is a waste of the bipolar output driver ICs. Also, the voltage for the output driver stage must be set to 5V for the servo inputs.

While the CANSERVO code uses the common pushbutton and LED sequence for allocation of a node number, it can only be configured in FLiM mode, using the facilities of the FCU or a similar software tool.

Node variables 36 (NV# of 1 to 36)

The first NV (NV1) is used for determining whether a servo output signal cuts off when the end position is reached or not. NV2 is used for testing the settings of a servo during the teaching process. NV3 and NV4 are currently unused.

The remaining NVs are in blocks of four, one block per servo output. Within each block, the first two NVs set the end positions for an ON and OFF event respectively and the second two NVs set the speed of travel corresponding to the ON and OFF events.

Number of stored events 32 or 128 depending on the firmware version (EN# of 1 to 32 or 128)

Number of EVs per event 2. (EV# of 1 or 2)

EV1 determines which servo output the event applies to. Allowed values of 00000000 to 11111111 where each bit corresponds to one of the 8 outputs. A bit set makes that output active. Multiple servos can move with a single event.

EV2 determines the polarity of each active servo output. Allowed values of 00000000 to 11111111. A bit set reverses the output relative to the ON or OFF OpCode.

The module ID number is 11. The manufacturer number is 165.

Teaching the CANSERVO module.

Give the module its FLiM Node Number by the usual process. The CANSERVO cannot be used in SLiM mode.

The CANSERVO will drive 8 conventional R / C servos, one from each output and numbered 1 to 8. Each servo can be set for its end positions and movement speeds. With a servo connected to the appropriate output, put the module into learn mode.

<0x53><NN hi><NN lo>

The positions and speeds are set using Node Variables (NVs) These start at NV5 and are in groups of 4. The sequence is:

NV(n) Position for an ON event NV(n + 1) Position for an OFF event NV(n + 2) Speed for an ON event NV(n + 3) Speed for an OFF event.
The configuration software device should send the teaching command to each NV using its index. (NVSET)

<0x96><NN hi><NN lo><NV#><NVval>

When setting the servo positions, the above command can be sent repeatedly, varying the NVval each time and the servo will follow the value. The centre value is 127 with the extremes of 0 and 255. The same process is followed for both the ON and OFF positions. There is no reason which way should be ON or OFF, i.e. the ON position can be clockwise and the OFF be anticlockwise or vice versa.

The speed values are set by the same process. However, the value range is 0 to 7 with 0 being the fastest (set only by the servo mechanism) and 7 is the slowest. If a speed is set, then subsequent tracking when setting the positions will be at the set speed. It is recommended that the speeds be set to 0 when setting the positions and then changed for the required speed. This makes setting the positions easier.

The effect of a setting process can be tested while in learn mode by sending a value to NV2.

The format is 'D000NNNN' where D is direction (1 is ON and 0 is OFF). NNNN is the servo number, 0001 to 1000 for servos 1 to 8. The servo will now act as if an event had been sent so you can check if the positions and speeds are satisfactory.

The above process can be repeated for each servo.

NV1 is used to set whether a servo output cuts off after about 2 seconds when the end position is reached or not. Some servos have been found to buzz or jitter when stationary and removing the drive signal prevents this. However, there is now no reference for the servo position so if the output shaft is moved by external forces, it will not try to drive back to its correct position. Hence the option in NV1. A one bit for each servo in NV1 sets the cutoff. The default is all set to all cutoff. (NV1 = 0xFF)

When the process is completed, take the module out of learn mode.

<0x54><NN hi><NN lo>

All NV settings can be read back using NVRD.

<0x71><NN hi><NN lo><NV#>

The response is NVANS.

Teaching events. This follows the same process as for the other consumer modules. There are 32 or 128 allowed events depending on the firmware version which can be either long or short. Each event has two EVs, EV1 sets which servo(s) the event applies to and EV2 the direction of that servo movement relative to an ON or OFF event.

Stored events and EVs can be read back using NERD or NENRD.

Supported OpCodes (HEX value and mnemonic) in FLiM mode

HEX	Mnemonic		HEX	Mnemonic		HEX	Mnemonic
			59	WRACK			
10	RQNP		5C	BOOTM			
						98	ASON
42	SNN		6F	CMDERR		99	ASOF
50	RQNN		70	EVNLF		9B	PARAN
51	NNREL		71	NVRD		9C	REVAL
52	NNACK		72	NENRD		B2	REQEV
53	NNLRN		73	RQNPN			
54	NNULN		74	NUMEV			
55	NNCLR					D2	EVLRN
56	NNEVN		90	ACON		D3	EVANS

57	NERD		91	ACOF		
58	RQEVN		95	EVULN	EF	PARAMS
					F2	ENRSP

Hardware considerations.

R/C servos consume current in pulses while driving to position. These pulses can be as high as 1 Amp per servo with a typical average of 200mA. While servos use a DC supply of 5 to 6V, it may not be wise to run them off the same 5V DC supply as is used by the CANSERVO module for its processor. The pulsatile nature of the current may interfere with correct operation of the processor. The MERG CANSERVO PCB and the CANACC8 servo adapter board allow for a separate DC supply for the servos.

When at the end positions, with the pulse train stopped, the current in each servo is steady and about 10mA per servo depending on the make and type.

CANSERVO8C (provisional)

The CANSERVO8C is a development of the CANSERVO that will run on a CANACC8 module or on the CANSERVO module. When used with the CANACC8 module, the output IC (U4), which is a Darlington driver, should be replaced with a resistor array (resnet) of 8 x 1K resistors. The 12V version of the CANACC8 (ACC8_2) has output options for either the Darlington array or the resnet. On this board, the jumpers for the outputs should be set to the resistor network. The same code can be used in the CANACC5 board but this is a waste of the bipolar output driver ICs. Also, the voltage for the output driver stage must be set to 5V for the servo inputs.

While the CANSERVO8C code uses the common pushbutton and LED sequence for allocation of a node number, it can only be configured in FLiM mode, using the facilities of the FCU or a similar software tool.

Node variables 37 (NV# of 1 to 37)

The first NV (NV1) is used for determining whether a servo output signal cuts off when the end position is reached or not. NV2 sets the position at startup. Each bit corresponds to one servo. A bit set will cause the servo to run to the OFF end and a bit clear will cause it to run to the ON end as set in the servo position NVs. NV3 is used in conjunction with NV2 to determine whether a particular servo moves at all on startup. A bit set will allow it to run and a bit cleared will give no movement on startup. NV4 determines whether a particular servo will move on a command event while other servos are running or wait till others have finished. A bit set will cause that servo to wait. A bit clear will allow servos to move together. This NV will sequence servo movement so power requirements are not excessive if several servos try to move at the same time. The remaining NVs are in blocks of four, one block per servo output. Within each block, the first two NVs set the end positions for an ON and OFF event respectively and the second two NVs set the speed of travel corresponding to the ON and OFF events. The NV with index of 37 is for testing during setup with the FCU.

Number of stored events 128 (EN# of 128)

Number of EVs per event 4. (EV# of 1 to 4)

EV1 determines which servo output the event applies to. Allowed values of 00000000 to 11111111 where each bit corresponds to one of the 8 outputs. A bit set makes that output active. Multiple servos can move with a single event.

EV2 determines the polarity of each active servo output. Allowed values of 00000000 to 11111111. A bit set reverses the output relative to the ON or OFF OpCode.

EV3 is used to allocate events for position reporting by the CANSERVO8C code. There are 4 such 'feedback' events allowed per servo.

EV4 is available but not currently used.

The module ID number is 19. The manufacturer number is 165. **Teaching the CANSERVO8C module.**

Give the module its FLiM Node Number by the usual process. The CANSERVO8C cannot be used in SLiM mode.

The CANSERVO8C will drive 8 conventional R/C servos, one from each output and numbered 1 to 8. Each servo can be set for its end positions and movement speeds. With a servo connected to the appropriate output, put the module into learn mode.

<0x53><NN hi><NN lo>

The positions and speeds are set using Node Variables (NVs) These start at NV5 and are in groups of 4. The sequence is:

NV(n) Position for an ON event NV(n + 1) Position for an OFF event NV(n + 2) Speed for an ON event NV(n + 3) Speed for an OFF event. The configuration software device should send the teaching command to each NV using its index. (NVSET)

<0x96><NN hi><NN lo><NV#><NVval>

When setting the servo positions, the above command can be sent repeatedly, varying the NVval each time and the servo will follow the value. The centre value is 127 with the extremes of 0 and 255. The same process is followed for both the ON and OFF positions. There is no reason which way should be ON or OFF, i.e. the ON position can be clockwise and the OFF be anticlockwise or vice versa.

The speed values are set by the same process. However, the value range is 0 to 7 with 0 being the fastest and 7 is the slowest. If a speed is set, then subsequent tracking when setting the positions will be at the set speed. It is recommended that the speeds be set to 0 when setting the positions and then changed for the required speed. This makes setting the positions easier.

The effect of a setting process can be tested while in learn mode by sending a value to NV37.

The format is 'D000NNNN' where D is direction (1 is ON and 0 is OFF). NNNN is the servo number, 0001 to 1000 for servos 1 to 8. The servo will now act as if an event had been sent so you can check if the positions and speeds are satisfactory.

The above process can be repeated for each servo.

NV1 is used to set whether a servo output cuts off after about 2 seconds when the end position is reached or not. Some servos have been found to buzz or jitter when stationary and removing the drive signal prevents this. However, there is now no reference for the servo position so if the output shaft is moved by external forces, it will not try to drive back to its correct position. Hence the option in NV1. A one bit for each servo in NV1 sets the cutoff. The default is all set to all cutoff. (NV1 = 0xFF)

NV2 is used to so servos will drive to a known position on power up. Each of the 8 bits corresponds to one servo, the LSbit is servo 1 and the MSbit is servo 8. A bit set (1) will drive the servo to the OFF end of travel and a bit clear (0) to the ON end of travel. Default is all moving to OFF end. (NV2 = 0xFF)

NV3 is used to override NV2 if you don't want the servo to move at all on power up. A bit set allows movement and a bit clear prevents movement. The default is all moving (NV3 = 0xFF)

NV4 is used to enable sequential movement of servos on the module. As servos take current in pulses of up to 0.5 amps peak when moving, the current if all 8 move at once could be excessive. A bit set in NV4 tells that servo to wait till others have stopped. Where several are requested to move, the servo with the lowest number will move first. In NV4, a bit set will cause that servo to wait. A bit clear will allow simultaneous movement. The default is sequential for all. (NV4 = 0xFF)

When the process is completed, take the module out of learn mode.

<0x54><NN hi><NN lo>

All NV settings can be read back using NVRD.

<0x71><NN hi><NN lo><NV#>

The response is NVANS.

Teaching events. This follows the same process as for the other consumer modules for the commands to move. . There are 128 allowed events which can be either long or short. Each event has two EVs for movement. EV1 sets which servo(s) the event applies to and EV2 the direction of that servo movement relative to an ON or OFF event. Note that CANSERVO8C also has EV3. For command events, this should be set to 0.

Stored events and EVs can be read back using NERD or NENRD.

This version has the ability to send back events when end positions are either left or reached and also an event when the mid travel position is crossed. These must be taught in a similar way to the command events. The difference is in the value of EV3. This must be set as follows.

ABCNNNDD

A indicates a feedback event. It must be set to 1. B indicates the polarity of the response. If set to 0, an ON event will be sent when the servo reaches the end of travel and an OFF event when it leaves that end. Setting it to a 1 will reverse the polarity. C sets the polarity of the mid travel event. If set to 0, it will send an ON event when travelling towards the ON end. If set to 1, it will give an OFF event. This mid travel event is intended to work in conjunction with a CANACC8 and a relay for frog switching. The ability to switch polarity may be useful if the frog changes incorrectly. NNN is the servo in question. 000 is servo 1, 111 is servo 8. DD sets which end or mid position the event applies to. 00 is for the ON end of travel, 01 is for the OFF end of travel, 10 is for the mid position. 11 is not presently used.

Each servo now has three events associated with it for position reporting. These can be either short or long events, depending on what was taught. Normally, they would be taught as short events (NNhi, NNlo = 00) so the end positions become 'device numbers'. These feedback events can be used to set the lights on a control panel, set signals etc and report to a PC for interlocking. They do not guarantee the turnout or signal has actually changed. It is a useful alternative if fitting position microswitches is not wanted.

When teaching the response events, you only need to teach with EV3 set. EV1 and EV2 are not relevant.

Notes:

If a command is sent to move a servo and it is already at that end, it will report back with an ON event for that end even though it hasn't moved.

If the servo end positions (Device numbers) are polled with a short request (ASRQ), they will reply with the corresponding state, as a short response.

This version of the PIC code includes the ability to force a CAN_ID self enumeration using OpCode ENUM (0x5D), to set a specific CAN_ID using CANID (0x75) and will self enumerate if the module pushbutton is pressed briefly.

Supported OpCodes (HEX value and mnemonic) in FLiM mode.

HEX	Mnemonic		HEX	Mnemonic		HEX	Mnemonic
0D	QNN		59	WRACK		96	NVSET
10	RQNP		5C	BOOTM		97	NVANS
11	RQMN		5D	ENUM		98	ASON
42	SNN		6F	CMDERR		99	ASOF
50	RQNN		70	EVNLF		9B	PARAN
51	NNREL		71	NVRD		9C	REVAL
52	NNACK		72	NENRD		B2	REQEV
53	NNLRN		73	RQNPN		B5	NEVAL
54	NNULN		74	NUMEV		B6	PNN
55	NNCLR		75	CANID		D2	EVLRN
56	NNEVN		90	ACON		D3	EVANS
57	NERD		91	ACOF		E2	NAME
58	RQEVN		95	EVULN		EF	PARAMS
						F2	ENRSP

Hardware considerations.

R / C servos consume current in pulses while driving to position. These pulses can be as high as 1 amp per servo with a typical average of 200mA. While servos use a DC supply of 5 to 6V, it may not be wise to run them off the same 5V DC supply as is used by the CANSERVO module for its processor. The pulsatile nature of the current may interfere with correct operation of the processor. The MERG CANSERVO PCB and the CANACC8 servo adapter board allow for a separate DC supply for the servos.

When at the end positions, with the pulse train stopped, the current in each servo is steady and about 10mA per servo depending on the make and type.

2. The DCC system.

CBUS incorporates a set of OpCodes for messages relating to loco control. Although primarily intended for implementation of DCC systems compliant with the NMRA Standards and RPs, it does not preclude use with analog or DC loco control systems given a suitable 'command station'. The controlling devices (CABs) or computer control along with a suitable command station, communicate via the CBUS wiring. CABs can also send layout control messages if required directly to the accessory modules as the bus is common. The following section describes the present MERG implementation of a DCC scheme using CBUS.

2.1 CABs

The CABs plug into the CBUS and appear as a node on the bus. However, CABs are considered generic in that they are all identical in functionality and are not programmable in the way the layout nodes are. CABs have been given a fixed node number of 0xFFFF for the purposes of 'bootloading' and reading properties. (See section on bootloading for details). This does not preclude CABs being allocated different node numbers if implemented differently. Except when controlling layout modules directly, CABs require a matching Command Station or a computer emulating a Command Station also connected to the CBUS. Loco control interactions take place between the CABs and the Command Station using a dedicated set of OpCodes.

Cabs may also issue CBUS short or long events for layout control, using whatever scheme the cab designer wishes.

==== 2.1.1 Operating sequence. ====

When first plugged in or powered up, a CAB is in its 'reset' state. It can receive CBUS messages but does not send DCC related messages. It may send layout control messages to devices (short events) if equipped to do so. Hence any CAB can activate the same layout 'device'. See section 2.5 for more details.

2.1.2 Allocation of a session.

When operating a loco, the CAB is allocated a 'session' number by the command station. In practice, it is a pointer to a 'slot' in the command station refresh table or 'stack'. This is a single byte so limiting the number of possible active CABs (or slots) to 256. If a CAB can control multiple locos, each loco is allocated a 'session'. To activate a session, the user needs to enter the loco address and send the following message (RLOC) to the command station.

<0x40><AAAAAAAA><AAAAAAAA>

The two bytes are the address of the loco requested. The address format is that used by the NMRA DCC scheme. If it is a 'short' address, the upper byte is all zeros as is bit 7 of the lower byte. Thus short addresses of 0 to 127 are allowed. An address of 0 should only be used if the command station allows for decoderless locos to be run on DC. For long addresses, bits 6 and 7 of the upper byte should be set by the CAB. This format allows for long addresses in the range 0000 to 3FFF although the NMRA specification only allows a range of 27FF (10239).

The number of active sessions will depend on the command station design and is really limited by the refresh rate required for the DCC packets to the track. Provided a session or slot is available, then command station will reply with PLOC.

<0xE3><Session><AddrHi><AddrLo><Speed/Dir><Fn1><Fn2><Fn3>

This comprises the full information of the 'slot' but tells the CAB its allocated session number.

For a new allocation, the speed / dir byte will be 0. If a moving train is being re-acquired, then the speed/dir byte will indicate the current speed and direction of the train."

If the command station remembers the function settings of previously controlled locos, these will be passed back to the cab in this message, otherwise they will be set to zero. The direction of the stationary loco can also be included, as this may affect which lights are illuminated.

If there are no available slots, the command station will give an error message ERR. Loco stack full.

<0x63><AddrHi><AddrLo><1>

If the loco with that address is already allocated to another session, either on a different CAB or on the same CAB if multiple locos are allowed, it will send the error message ERR. Loco address taken.

<0x63><AddrHi><AddrLo><2>

To release a loco from a session use KLOC.

<0x21><Session>

This clears the slot and makes it available for other locos. If a loco is released whilst moving, this is referred to as "Dispatched". The train keeps moving and continues to be refreshed by the command station. It can then be re-acquired by a cab by issuing RLOC as described above.

A cab must monitor for ERR messages and cancel the current session if it receives an ERR message on its current session of either "No session" or "Session cancelled".

If the cab receives a “loco taken” error message, it may then “Steal” or “Share” the session.

To do this, the cab issues a GLOC (Get Loco) message to the command station:

<0x63><AddrH><AddrL><Flags>

The flags byte is defined as:

Bit 0: Set for “Steal” mode - The cab wants to steal the session from the cab(s) that currently own it, so their sessions will be canceled

Bit 1: Set for “Share” mode - The cab wants to share the session with the cab(s) that currently own it.

Both bits set to 0 is exactly equivalent to an RLOC request Both bits set to 1 is invalid, because the 2 modes are mutually exclusive

Possible returned packets from the command station are as follows:

PLOC - the request was successful, the PLOC packet contains the session number, speed/direction and function settings as described above in the response to RLOC.

ERR with error code 2 (loco taken) - Requested operation (Steal or share) is disabled or not supported

ERR with error code 3 (no session) - There is no existing session on that loco to steal or share

ERR with error code 7 (invalid request) – There is an invalid combination of flags, i.e: both steal and share are set.

When the command station accepts a steal request, it will first issue an ERR message on the session in question with the error code 8 “Session Cancelled".

This indicates that the current session is being stolen. Cab(s) with that session must cancel the session. A suitable message can be displayed to the user. The cab doing the stealing should ignore this error code whilst waiting for a PLOC.

GLOC was introduced with version 8a of the CBUS specification. Cab and command station designers should bear in mind that their design should also work with older Cabs/command stations that do not implement GLOC to ensure backwards compatibility.

Any cab implementing the new OpCode GLOC must have a timeout in case no response is received.

This is so that if such a cab is used with command station firmware (that does not yet support GLOC), then it won't all lock up if you try to use Steal or Share. If a GLOC request times out, a suitable message can be displayed to the user.

Any command station implementing GLOC must continue to support RLOC as well.

Sharing sessions implies a need for each cab to monitor activity from the other so that it is aware of current speed/direction and function settings for the shared session. A mechanism for speed matching between the cabs that are sharing a session may also be required. All of the information required for this is available in the CBUS messages, but it is left to the cab designer to implement suitable mechanisms on the cab user interface.

2.1.3 Setting speed step range.

Provided the session is granted, (PLOC) then the CAB should send a message giving the speed step range required for that loco. This is done with a STMOD.

<0x4d><Session><xxxxxxDD>

Only the last two bits of the data byte are used. See specification section for values. It is recommended that the command station defaults to 128 speed steps. Unless the loco requires a different range, the STMOD message may be omitted.

2.1.4 Keepalive.

Once a session has been granted, the CAB must send regular ‘keepalive’ messages DKEEP. This enables a command station to recognize if a CAB has been disconnected and the keepalive should be at least once every 4 seconds.

<0x23><Session>

2.1.5 Speed and direction.

Once a CAB has its session, it can send speed / direction and function messages to the command station. The speed / dir is a byte where bits 0 – 6 are the speed and bit 7 is the direction. 1 is forward and 0 is reverse. The speed / direction is DSPD.

<0x47><Session><Speed/Dir>

Note that any valid speed / direction message to that session should also act as a keepalive and reset any command station timer. Also, the speed format follows that of the NMRA DCC where speed 1 is considered as emergency stop. For normal running, a CAB should send speeds of 0, 2, 3 .. . etc. It is recommended that when speed is being changed, the rate of sending DSPD messages is limited to prevent congestion on the bus. This rate is a compromise between too many bus messages and too sluggish a loco response. Experience with the MERG system which has 32 possible locos active (32 slots in the stack) shows that a 32 millisecond interval between speed change messages from any one CAB is satisfactory.

2.1.6 Function control.

There are presently two methods of setting functions. The first, using DFUN, is the one currently implemented by the MERG command station and follows the NMRA DCC function byte protocol. The second (new) proposal uses DFNON and DFNOF.

With DFUN, the CAB sends the function byte in the DCC format. This makes life easier for the command station.

<0x6b><Session><FR><Fn byte>

Here FR is the function range as shown in the specification section and Fn byte is the DCC formatted function command byte for that range. This CBUS message will be sent for any requested change in a function setting. In accordance with the NMRA recommendations, functions F0 to F12 should be regularly refreshed while functions F13 to F28 may be sent once only. The information returned in a PLOC or QLOC message will contain the three function control bytes covering ranges F0 to F4, F5 to F8 and F9 to F12 as these will normally be contained in the stack.

The use of DFNON and DFNOF simplifies the setting of functions by the CAB or computer but needs additional processing by the command station. The format is very simple.

DFNON (Function ON)

<0x49><Session><Fnum>

DFNOF (Function OFF)

<0x4A><Session><Fnum>

As Fnum is a byte it allows for 256 possible functions but only F0 to F28 are presently defined by the NMRA.

As presently implemented, the MERG CABs allow for any function to be set as either ON / OFF or ‘Momentary’. While in loco driving mode, if a Fn button is pressed and the ‘Consist’ button is pressed together, the mode on the screen with show either ON / OFF or MOM. Repeated pressing of both buttons together toggles between modes. In the ON / OFF mode, the display shows the Fn number and either ON or OFF with each press. In momentary mode, it shows MOM+ if the button is held down and MOM- when released. This operating mode is retained within the CAB and not held in the Command Station.

2.1.7 Emergency stop.

The CBUS protocol has a request emergency stop all OpCode, RESTP. . This can be used to stop all locos via the command station. To initiate an emergency stop all, a cab (or PC software) should issue the single byte RESTP message:

<0x0A>

The command station should respond by issuing the broadcast stop DCC packet to the track, and sending the track stopped ESTOP single byte message on CBUS:

<0x06>

All other cabs should then respond by displaying a suitable message to the user and issuing no more non-zero speed packets until stop all is cleared for that train. Keep alive messages should continue to be sent. On the MERG cabs this requires the user to reset the knob to zero which then allows the train to be restarted from rest. A cab with an encoder or a software throttle such as JMRI can simply set

the displayed speed to zero and then similarly allow the user to restart from rest.

However, we have found that the ability to stop just 'your' loco is useful. This is simply achieved by sending a speed / dir message (DSPD) with the speed set to 1. The MERG CANCMD and CAB combination utilizes a single press of the Em. Stop to stop your loco and a double press to stop all locos.

In addition to the command station receiving the ESTOP, all other active CABs should recognize this and act appropriately, e.g. cease sending DSPD messages and indicate a 'stop all' has been issued.

2.2 Consisting.

The CBUS protocol has OpCodes for 'advanced consisting' where the loco is put into consist mode by writing the consist address to CV19. OPC is PCON.

<0x45><Session><Consist#>

The session is that of the loco to be put into the consist. Consist# is the 7 bit consist address with bit 7 indicating the direction in the consist. Bit 7 set reverses the direction.

To run the consist, you establish a session for the consist address as if it were a loco with a short address. The NMRA advanced consisting principle does not allow you to control a single loco and the consist separately if both have the same short address. The CBUS specification also has the OpCode KCON for removing a loco from a consist but this functions identically to PCON but with the Consist# set to 0. You need to establish a session for the loco by its own address, not that of the consist.

<0x46><Session><0>

By using PCON, you can change a loco from one consist to another without using KCON.

Note: For anyone wishing to use the so called 'universal consisting' where locos are sent individual speed and direction packets in sequence, the command station needs just to know which locos in its stack are also in a consist and send the appropriate speed and direction packet to each. In this case, PCON can be interpreted by the command station in a different way. It is not practicable to run advanced consisting and universal consisting at the same time.

2.3 CV programming.

The CBUS specification has OpCodes to allow all the NMRA specified programming modes. This includes On The Main (OTM) programming and full 'service mode' programming and CV readback. Currently, readback OTM using systems like RailComtm has not been implemented but CBUS would be very suitable as a 'feedback' bus from any track detector scheme.

OTM programming uses WCVO for byte write and WCVB for bit mode write. For byte write

<0x82><Session><High CV#><Low CV#><Val>

The session is that of the loco to be programmed. The two CV bytes allow for 65535 possible CVs but only 1023 are allowed by the NMRA. Val is a byte to be entered into the CV. For bit mode write, use WCVB

<0x83><Session><High CV#><Low CV#><Val>

Here Val is a byte as defined in RP 9.2.1 for OTM bit manipulation in a DCC packet.

2.3.1 Service mode.

Service mode read and write of CVs are allowed in direct, page, register and address only modes.

To write a CV in service mode use WCVS

<0xA2><Session><High CV#><LowCV#><Mode><CVval>

As a loco in service mode will be on the programming track, it need not have a specific address. Thus the 'session' is only used to link a CAB or programmer software to the command station / programming device. Any 'session' will do. You can continue running a loco on the main and use its 'session' to program a different loco on the programming track. The 'mode' byte defines the service mode to be used and CVVal is the value to be written, or, if in 'bit manipulation' mode, the appropriate bit pattern as defined in RP 9.2.2. The modes are shown in the specification section but are repeated here for convenience.

0 Direct Byte 1 Direct Bit 2 Page Mode 3 Register Mode 4 Address Only Mode

A command station / programmer may not support all modes.

Following the WCVS message, the command station / programmer will respond with SSTAT.

<0x4C><Session><Status>

The status indicates the result of the write process. Again this is defined in the specification section but repeated here.

0 Reserved 1 No Acknowledge 2 Overload on service mode programming track 3 Write Acknowledge 4 Busy 5 CV out of range

Reading a CV follows the same process. The OpCode is QCVS.

<0x84><Session><High CV#><Low CV#><Mode>

A successful read results in a PCVS.

<0x85><Session><High CV#><Low CV#><Val>

Where Val is the value of the CV as a HEX byte. If the read process fails, a SSTAT message will be sent by the programmer with the reason for failure in the 'status' byte.

2.4 Direct transmission of DCC packets.

CBUS also has a set of OpCodes which allow a 'device', usually a PC, to act as a command station and send messages in direct NMRA DCC format. The track interface will now simply take these messages and convert them directly into a DCC packet with the correct bit format and timing. These OpCodes are RDCC3, RDCC4, RDCC5 and RDCC6. The minimum DCC packet has three bytes and the maximum had six bytes, including the error byte. The OpCode also includes a byte specifying the number of times the DCC packet is to be sent.

2.5 Layout control from CABs.

It may be useful to control accessories on the layout directly from CABs. This simply requires CABs to send the appropriate CBUS event. However, CABs are 'generic' in that they are all essentially the same and so have no unique Node Number. By default, all cabs have the same constant node number of 0xFFFF. This means that any 'long' events sent from a cab will always have the cab node number 0xFFFF. Therefore long events are not really suitable for general purpose layout control from a cab, instead short events are more suitable as the device number used can refer to the device being controlled. Long events are, however, suited to events issued from a cab that are directly related to a cab operation.

As presently implemented, the MERG CBUS CABs send a 'short' event for layout control where the user enters the device number of the accessory to be changed. The implementation used is as follows: (From firmware Rev 2r onwards)

Provided that the CAB is not 'busy' doing things expecting an 'Enter' press, e.g. programming CVs, then pressing Enter alone gives a screen prompt for accessory operation. This is 'AC=' . The user enters the device number with the numeric keys. A press of the 'Consist' button will send alternate ON and OFF events to that device. The display shows a '+' or a '-' corresponding to ON or OFF. The loco address and speed remain displayed and the loco is under full control during any accessory setting process. To change the accessory number, simply enter that number with the buttons. If you enter a wrong number, press Enter. This will clear the existing number and allow a new one to be entered. You can operate accessories with no loco selected and from any CAB. If the same accessory is changed by one CAB or other means, for example with a PC, the sign of the display will change on all CABs that have that accessory selected.

During accessory operations, the numeric buttons cannot be used for 'Functions'.

The accessory device number range is limited to between 0 and 9999, purely by the display size. Accessory mode is cancelled just by pressing the 'Loco' button again.

The MERG cab also sends a long event when a loco function key is pressed, in addition to the usual DCC function message sent to the command station. The event number corresponds to the function button pressed.

This allows a user to teach such events to consumers that carry out an action specifically related to a loco function selection. The node number is always set to 0xFFFF so all consumers will respond the same regardless of which cab the event comes from.

A couple of examples: If F3 always works a horn, the effect can be enhanced by teaching the F3 cab CBUS event to a consumer that sounds a horn via a larger speaker under the layout. Thus non-sound fitted locos would still get a horn noise. A coupling noise can operate a layout uncoupler at the same time.

A user can also teach events from function buttons that are not used for their locos as quick access layout controls, such as often used routes. For example, a user who is not using sound is unlikely to use any loco functions above F4 or so, leaving F5 upwards available for quick access layout control.

The use of these events is entirely up to the user, and can be ignored if the user does not desire to use this feature.Obviously, this is a specific implementation and other CAB designs may use different sequences.

Mike Bolton 01/09/2013 ©
