**CANID\_UNI\_BMS\_I/R**

Battery module UNIversal BMS node CAN msg.

The **I** suffix (Incoming msg to BMS node) provides commands originated by the following CAN msgs, generally assigned to the EMC (Energy Management Control) function, and PC, (see CAN ID assignment file: ~/GliderWinchCommons/embed/svn\_common/trunk/db/CANID\_INSERT.sql)

INSERT INTO CANID VALUES ('CANID\_UNI\_BMS\_EMC\_I','B0000000', 'BMSV1', 1,1,'U8\_U8\_U8\_X4','BMSV1 UNIversal From EMC, Incoming msg to BMS: X4=target CANID');

INSERT INTO CANID VALUES ('CANID\_UNI\_BMS\_PC\_I' ,'B0200000', 'BMSV1', 1,1,'U8\_U8\_U8\_X4','BMSV1 UNIversal From PC, Incoming msg to BMS: X4=target CANID');

Program loading

Request sending Cell voltage readings

Request sending various readings

Request setting BMS modes (e.g. dump discharge)

The **R** suffix (Response msg)--

The following are responses to all commands plus heartbeat timeouts. The **same** CAN ID is used for all responses. The code in payload [0] designates the format of the remainder of the payload. All msgs generated by the BMS node use the CAN ID assigned to the CAN loader for the processor on the node. The only node that will generate CAN msgs with this CAN ID is the PC CANloader program. These msgs carry payload[0] codes associated with loading programs over CAN and the sending/response is such that CAN msgs generated by the two nodes will not conflict.

The CAN ID associates with a BMS module via the parameter file <aaaaaaaa>-bq\_idx\_v\_struct.c,

e.g. B0A00000-bq\_idx\_v\_struct.c--

In directory: ~/GliderWinchItems/BMS/bmsadbms1818/params

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV11R','B0201114', 'BMSV1', 1,1,'U16\_U16\_U16\_U16','01 BQ76952 #BQ01 U16:id,U16:cellv n+1,U16:cellv n+2');

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV12R','B0201124', 'BMSV1', 1,1,'U16\_U16\_U16\_U16','02 MAX14921 #MX01 U16:id,U16:cellv n+1,U16:cellv n+2');

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV13R','B0201134', 'BMSV1', 3,1,'U16\_U16\_U16\_U16','03 ADBMS1818 #AD02 U16:id,U16:cellv n+1,U16:cellv n+2');

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV14R','B0201144', 'BMSV1', 4,1,'U16\_U16\_U16\_U16','04 ADBMS1818 #AD03 U16:id,U16:cellv n+1,U16:cellv n+2');

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV15R','B0201154', 'BMSV1', 5,1,'U16\_U16\_U16\_U16','05 ADBMS1818 #AD04 U16:id,U16:cellv n+1,U16:cellv n+2');

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV16R','B0201164', 'BMSV1', 6,1,'U16\_U16\_U16\_U16','06 ADBMS1818 #AD05 U16:id,U16:cellv n+1,U16:cellv n+2');

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV17R','B0201174', 'BMSV1', 7,1,'U16\_U16\_U16\_U16','07 ADBMS1818 #AD06 U16:id,U16:cellv n+1,U16:cellv n+2');

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV18R','B0201184', 'BMSV1', 8,1,'U16\_U16\_U16\_U16','08 ADBMS1818 #AD07 U16:id,U16:cellv n+1,U16:cellv n+2');

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV21R','B0201214', 'BMSV1',11,1,'U16\_U16\_U16\_U16','09 ADBMS1818 #AD08 U16:id,U16:cellv n+1,U16:cellv n+2');

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV22R','B0201224', 'BMSV1',12,1,'U16\_U16\_U16\_U16','10 ADBMS1818 #AD09 U16:id,U16:cellv n+1,U16:cellv n+2');

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV23R','B0201234', 'BMSV1',13,1,'U16\_U16\_U16\_U16','11 ADBMS1818 #AD10 U16:id,U16:cellv n+1,U16:cellv n+2');

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV24R','B0201244', 'BMSV1',14,1,'U16\_U16\_U16\_U16','12 ADBMS1818 #AD11 U16:id,U16:cellv n+1,U16:cellv n+2');

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV25R','B0201254', 'BMSV1',15,1,'U16\_U16\_U16\_U16','25 BMSV1 U8:n,U16:cellv n,U16:cellv n+1,U16:cellv n+2');

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV26R','B0201264', 'BMSV1',16,1,'U16\_U16\_U16\_U16','26 BMSV1 U8:n,U16:cellv n,U16:cellv n+1,U16:cellv n+2');

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV27R','B0201274', 'BMSV1',17,1,'U16\_U16\_U16\_U16','27 BMSV1 U8:n,U16:cellv n,U16:cellv n+1,U16:cellv n+2');

INSERT INTO CANID VALUES ('CANID\_MSG\_BMS\_CELLV28R','B0201284', 'BMSV1',18,1,'U16\_U16\_U16\_U16','28 BMSV1 U8:n,U16:cellv n,U16:cellv n+1,U16:cellv n+2');

Preferred address assignment layout: 29b ID--

bits 1-11-- message type

bits 12-13-- “master” or sender (e.g. BMS, HC, PC, spare)

bits 14-29-- mfr & serial number

**PAYLOAD LAYOUT**

**Initiator** of command:

EMC or PC sends and module(s) respond. Responses can be for all, specified string, specified string and module, specific CAN ID.

**payload [0] U8: Universal code**

For the databast list of code numbers versus names see:

GliderWinchCommons/embed/svn\_common/trunk/db/CMD\_CODES\_INSERT.sql

**payload [1]** U8: conditional on Universal Code number

**If [0] code** == LDR\_\* (code numbers 1-24) are program loading commands

payload [1]-[7] detailed in loader program

**If [0] code** == CMD\_CMD\_CELLPOLL (**code 42**) Request to send cell readingsCMD\_CMD\_CELLPOLL

**Initiator**

payload[1] U8: Module identification

[7:6]

11 = All modules respond

10 = All modules on identified string respond

01 = Only identified string and module responds

00 = spare; no response expected

[5:4] Battery string number (0 – 3) (string #1 - #4)

[3:0] Module number (0 – 7) (module #1 - #16)

payload[2]

[7:4] ADC rate code (0-7) See BMSTask.h

[3:0] reading group sequence number

payload[3] - reserved

payload[4]-[7] CANID of unit to requested to respond

NULL = all;

Not NULL: expect a valid CAN ID.

**Response**s (BMS node might respond depending on initiating msg)

**If [0] of incoming code** = CMD\_CMD\_CELLPOLL (**code 42**)

Or, if heartbeat timeout: Send all CELL readings for the module.

**Response payload--**

payload[0] =

If heartbeat timeout: CMD\_CMD\_CELLHB (44) timeout

If emc sent CELLPOLL: 'CMD\_CMD\_CELLEMC (46 )

If pc sent CELLPOLL: CMD\_CMD\_CELLPC (47)

payload[1]

[7:4] “n” Cell number of first payload cell reading

[3:0] Reading group sequence number

payload[2-3] U16 - Cell n+0 reading (0.1 mv)

payload [4-5] U16 – Cell n+1 reading (0.1 mv) (conditional on DLC)

payload [6-7] U16 – Cell n+2 reading (0.1 mv) (conditional on DLC)

**If [0] code** == **CMD\_CMD\_TYPE2 (43)** Request specific BMS responses

**Initiator**

payload [1] U8: Sub-Code for request

[See cancomm\_items.h]--

// payload [1] U8: TYPE2 Command code

#define MISCQ\_HEARTBEAT 0 // reserved for heartbeat

#define MISCQ\_STATUS 1 // status

#define MISCQ\_CELLV\_CAL 2 // cell voltage: calibrated

#define MISCQ\_CELLV\_ADC 3 // cell voltage: adc counts

#define MISCQ\_TEMP\_CAL 4 // temperature sensor: calibrated

#define MISCQ\_TEMP\_ADC 5 // temperature sensor: adc counts for making calibration

#define MISCQ\_DCDC\_V 6 // isolated dc-dc converter output voltage

#define MISCQ\_CHGR\_V 7 // charger hv voltage

#define MISCQ\_HALL\_CAL 8 // Hall sensor: calibrated

#define MISCQ\_HALL\_ADC 9 // Hall sensor: adc counts for making calibration

#define MISCQ\_CELLV\_HI 10 // Highest cell voltage

#define MISCQ\_CELLV\_LO 11 // Lowest cell voltage

#define MISCQ\_FETBALBITS 12 // Read FET on/off discharge bits

#define MISCQ\_DUMP\_ON 13 // Turn on Dump FET for no more than ‘payload [3]’ secs

#define MISCQ\_DUMP\_OFF 14 // Turn off Dump FET

#define MISCQ\_HEATER\_ON 15 // Enable Heater mode to ‘payload [3] temperature

#define MISCQ\_HEATER\_OFF 16 // Turn Heater mode off.

#define MISCQ\_TRICKL\_OFF 17 // Turn trickle charger off for no more than ‘payload [3]’ secs

#define MISCQ\_TOPOFSTACK 18 // BMS top-of-stack voltage

#define MISCQ\_PROC\_CAL 19 // Processor ADC calibrated readings

#define MISCQ\_PROC\_ADC 20 // Processor ADC raw adc counts for making calibrations

#define MISCQ\_R\_BITS 21 // Dump, dump2, heater, discharge bits

#define MISCQ\_CURRENT\_CAL 24 // Below cell #1 minus, current resistor: calibrated

#define MISCQ\_CURRENT\_ADC 25 // Below cell #1 minus, current resistor: adc counts

#define MISCQ\_UNIMPLIMENT 26 // Command requested is not implemented

#define MISCQ\_SETFETBITS 27 // Set FET on/off discharge bits

#define MISCQ\_SETDCHGTST 28 // Set discharge test with heater fet load (more to be added)

payload [2] U8: ????

payload [3] U8: item number

for command codes 2 & 3: cell number (1-18)

for command codes 4 & 5: thermistor number (1-3)

payload[4]-[7] CAN ID of intended responder

NULL = all respond, otherwise the unit matching this CAN ID responds

**Response**s

payload[0] – [3] return bytes received from Initiator

payload[4]-[7] X4: requested response data (four bytes: bytes, integers, or float)

**If [0] code** == [0] code: CMD\_CMD\_TYPE3 (44)

payload[1]-[7] Reserved

**Heartbeat**

A BMS unit sends a CAN msgs periodically if no CAN msgs that request a response from that unit during the time between heartbeats.

Burst of 6 CAN msgs (3 cells per CAN msg = 18 cells)

Status msg:

[0] CMD\_CMD\_TYPE2 (43):

[1] MISCQ\_STATUS (1)

[4] Battery status bits (see bqTask.h)

[5] FET status bits (see bqTask.h)

[6-7] FET discharge bits

**Narrative**

The goal is to have a single CAN ID per BMS battery module. However, this cannot be achieved if the rule is followed that all senders of CAN msgs use a unique CAN ID. However, the number of different CAN IDs required can be minimized by having a flexible payload structure so that one CAN ID can convey all the needed data with just one CAN ID. This reduces the number of CAN IDs required to the number of senders.

At minimum there is just one sender. Such a case is a BMS unit that sends an unsolicited msg, e.g. a heartbeat or alert. However, the main use of the BMS is to send cell readings and those msgs are solicited by another CAN node, which would most likely be the EMC (Energy Management Controller). Others may solicit responses from the BMS, e.g. the PC loading a program. To avoid CAN ID conflict, a CAN ID for each sender is required. Hence, a minimum of three CAN IDs are needed: BMS, EMC, PC. Others are also likely, e.g. HC, and remote access.

Using a bit field structure for assigning the CAN IDs makes it easier to recognize CAN msg IDs manually without resorting to a table lookup. There is nothing in this scheme, however, that requires the CAN IDs to conform to the address layout shown above. The selection of CAN IDs is an administrative issue.

The BMS unit parameter lists only have their assigned CAN ID for sending, plus CAN IDs for recognition of incoming msgs from the EMC, PC, and other senders of interest. They do not have identification of the battery string they are on, and battery modules might be swapped between battery strings, winches, or inventory. To avoid reprogramming parameters lists when battery modules are changed, the battery string identification is done with a discovery process that is based on control of the master-reset lines.

The master-reset line is individually controllable for each battery string. The controlling node, most likely the EMC, can activate the master-reset for a battery string, which causes all the processors on that string to be held in reset.

To relate the battery modules to the string to which they are installed, the EMC first sends a msg that causes all battery modules on the bus to respond. The next step is to hold one string in reset and send msg again. The missing CAN IDs are the modules on the battery string that was being held in reset. This process is repeated for each of the battery strings. The result is that the EMC can build a list of CAN IDs versus battery strings without manual intervention.

An alternate procedure would be to hold all but one of the master-reset lines in reset and the reporting CAN units would be the list for that battery string.

Sending CAN msgs that are to be recognized by only one node/processor, but using only one CAN ID for the EMC, PC, etc., requires a way for a BMS unit to determine if an incoming CAN msg from the sender is specific to that unit. The BMS knows the CAN ID it uses for sending. The EMC, or PC, can specify the unit to respond by including the CAN ID of that unit in the payload of the command. The unit with a CAN ID matching the payload CAN ID responds to the command. A NULL CAN ID in the payload signifies that all BMS units respond to the command.

Putting the CAN ID in the payload takes four bytes which takes up half of the payload. Since at least one byte for a command payload is required, only three bytes remain for other data. However, to elicit a response from a specific BMS unit the command does not need to carry much information beyond identification so using half the total payload for the CAN ID not a problem. Sending the command codes and CAN ID are sufficient. The response, however, does not need to repeat the CAN ID since the CAN msg itself supplies the identification, so the response has most of payload usable for data. The program loading process is different as the bulk of data is being moved in the other direction, i.e. from the PC to the BMS unit.

For program loading, to specify the BMS unit to receive the data, several approaches can be used. The PC knows a priori the CAN ID of the node to reprogrammed. The issue is for the node to recognize that it is the intended recipient of the msg. As noted above including the CAN ID takes up four bytes of payload and at minimum one more is required for the command code. This leaves three bytes for data. Using this approach a 100K program will require, very roughly, ½ minute.

Another approach that could be used is based on breaking the rule that CAN IDs on msgs are unique to a node. For program loading, the node being reprogrammed is in a strict query/response mode, therefore the same CAN ID can be used for both the sender and responder without conflict.

To do this, the PC sends a msg using its unique sender CAN ID with the command code and CAN ID of intended recipient in the payload. The intended unit does a software triggered reset which starts the loader program. The loader program waits for a short duration for loader msgs using its own CAN ID. (The units with the loader installed, run the loader program when booting up after a reset. If no loading msgs are received after a short delay the application program is started.) The loader program ignores msgs that do not have loader commands. The PC then sends loader msgs using the recipient’s CAN ID and awaits acknowledgements, etc. Missed msgs and timeouts present a possible conflict so if the PC times out waiting for an acknowledgment it must resume the process using the PC’s unique CAN ID with the requisite command code and recipient CAN ID. This process frees up the payload so that seven of the eight bytes can be used in for loader data.

Either of the above procedures can be implemented with the CAN msg layout detailed above.

**PAYLOAD[0]**

In all the BMS CAN msgs the first byte, payload[0], carries a “universal” command code, which is a code used for all BMS modules nodes. The code numbers are assigned in the sql database, found in--

GliderWinchCommons/embed/svn\_common/trunk/db/CMD\_CODES\_INSERT.sql

A script in that directory creates a file containing #define statements that relate names to code numbers. It also creates a similar file for use by a java programs.

**CMD\_CMD\_TYPE1**

This payload format is for sending module cell readings.

Readout of the cells on a module can require as many as six CAN msgs to transmit the readings from a module with 18 cells. Payload[1] upper four bits specifies the cell number minus one (i.e. cells #1-#16 would be 0-15) of the first reading (“n”) which is a U16 in payload[2]-[3]. The size of the DLC determines if there are one, two, or three consecutive numbered cell readings. The sixth CAN msg for a battery module with 18 cells would have a “n” of 15, and a DLC of 8, yielding readings for cells #16, #17, #18. With a 16 cell module the “n” would 15 and a DLC of 4, yielding one reading for cell #16. Etc.

The payload cell voltages are 16b unsigned integers, calibrated in 0.1mv units. A payload of zero represents no reading, zero, or negative values. (The BQ76952 can report small negative voltages. The ADBMS1818 only reports positive values.)

To assure the CAN msgs for a readout are grouped correctly, a 4 bit sequence number is included. The node initiating the response sets the sequence number to be used by the BMS node or nodes that respond. The sequence number provides a check on missing CAN msgs and assures that the readings all resulted from the same command.

The node initiating the sending can send a command msg with either the CAN ID payload bytes set to NULL in which case all BMS modules will respond, or set to the BMS CAN ID in which only the BMS node with a CAN ID will respond. There is no simple way to restrict the responses to just the modules on one string since the modules do not know the battery string to which they are attached.

If readings from a specific string are needed, plus it is important to exclude the other strings from responding, then a scheme for setting the battery string in each module upon startup, could be devised. E.g., after the CAN IDs for a battery string have been identified using the discovery process described above, each node could be sent a command carrying the battery string number. The above payload format for commands would be used and a command codes assigned for this purpose.

**CMD\_CMD\_TYPE2**

This payload format is for commands**.**

Payload[0] carries the command code that is also used for the loader and other uses. When it is CMD\_CMD\_TYPE2 the units that are monitoring for CAN IDs from the EMC or PC, will respond according to the code that is in payload[1]. Since there are number of cases where the same commands are used to read the same type of reading, e.g. temperatures, from multiple sensors, or cell raw ADC counts, payload[2] is used for that purpose.

Payload[3] is spare/reserved.

Given further development experience, payload[3] might be designated a sequence number. Rather than send three commands to read three temperature sensors, the BMS units could respond with three CAN msgs, much the same as for cell readings. In this case the sequence number would be used to assure readings were grouped together, and the item number would identify the particular sensor, or item.

The responder(s) return the bytes received in payload[0]-[3].

Payload[4]-[7] holds a CAN msg that is **Initiator** uses to designated the unit to respond to the command. If this CAN ID is NULL all units respond, e.g. read the first temperature sensor from all battery modules with a single command. If the CAN ID is not NULL the unit with the matching CAN ID responds.

Payload[4]-[7] holds a four byte value that the **Responder**(s) use. This is an “X4” designation that can be signed or unsigned four bytes, two 16b, or one 32b, or float. The command determines the expected data.

Timeouts and heartbeats--

Normally, the EMC sends a command that requests all the BMS nodes to take cell readings and respond with the CAN msgs. If EMC msgs are not received after ‘x’ seconds the BMS node sends the readings as a heartbeat type of response.