**EIA-232 serial protocol to the QBridge**

The EIA-232 serial protocol to the QBridge is a simple, binary protocol. This document discusses the packet structure of a communication packet. Acknowledgements and retries needed and the format of the various commands which may be sent.

The Qbridge serial protocol is a stop-and-wait protocol. This means that the transmitting side is required to wait for an acknowledgement (ACK) to a packet sent before it may transmit the next packet. If no ACK is received within a reasonable time, the transmitter should retransmit the previous packet. A stop-and-wait protocol is generally easier to develop and maintain than a full streaming protocol, at the cost of lower throughput of data. In this case, since the EIA-232 communication link runs substantially faster than the J1708 bus, lower throughput on the EIA-232 link will have no effect on overall data transfer performance.

**Packet Structure**

Communication with the QBridge will be in the form of defined packets. Each packet will be ACKed upon receipt. A QBridge packet has the following structure:

| | |
|---|---|
| 1 byte | STX character (ASCII 0x02) |
| 1 byte | Total packet length. This includes the STX, length and CRC bytes. |
| 1 byte | Command (packet type). See the section Packet Types for a description of the various packet types and their data sections. |
| 1 byte | Packet ID. This value is used to detect duplicate packets. This value should be incremented by one for each packet that is sent. * This Packet ID should be incremented by one for each packet that is transmitted |
| Length less 6 bytes | Packet Data – the interpretation of this data is dependent on the command. |
| 2 bytes | CRC – 16 CITT CRC checksum (standard CRC used by most QSI products) |

* PacketIDs are generated on the transmitting device and checked for duplicates on the receiving device. Each device attached to the serial link should use an independent PacketID for packets that it generates. In other words, a device attached to the Qbridge does not need to worry about the PacketIDs that the Qbridge uses.

Due to the way that error recovery works, it is recommended that you not use a PacketID of zero.

**Error Recovery**

Serial communication links are inherantly unreliable and, while rare, it is expected that invalid packets will be occasionally be received. This section discusses what to do in those cases.

*Invalid Packets*

In the event that data is received that does not appear to be a valid packet (invalid length, invalid CRC), the receiving device should transmit an ACK packet with an ACK code of

ACK_INVALID_PACKET.

The receiving device should scan the serial stream for the next STX character in the serial stream which follows the STX character for the invalid packet, and resume packet parsing at that point.

*Incomplete Packets*
If a device receives an incomplete packet, and no more bytes are received for at least 100 ms, then the device should purge the incomplete packet up to the next STX character (or the entire receive buffer if no STX character exists) and transmit a ACK with the code of ACK_INVALID_PACKET.

*Duplicate Packets*
If a duplicate packet is received, the device should transmit an ACK packet with an ACK code of ACK_DUPLICATE_PACKET. The device is not required to re-process this packet. A duplicate packet is defined as one whose PacketID field matches the PacketID of the most recent, successfully received packet. A device is not required to track more than the most recent PacketID.

*Invalid Command*
If a packet is received with an invalid command, the device shall respond with an ACK packet with an ACK code of ACK_INVALID_COMMAND.

*Valid packets with erroneous data*
If a command is received with erronious data, the device shall respond with an ACK packet with an ACK code of ACK_INVALID_DATA

If a device does not receive an ACK from the Qbridge within a reasonable period (e.g. one second), or receives a ACK with the parameter of ACK_INVALID_PACKET, the device should attempt to retransmit the last command sent. If the Qbridge fails to respond or reports a ACK_INVALID_PACKET three consecutive times, the host device may assume that the Qbridge is not present or has failed.

**Commands**

The following commands are defined, along with the data that is associated with them.

| Command | Byte | Description |
|---|---|---|
| Init | 0x40 '@' | Initializes the communication link with the Qbridge. A device should always transmit this command before beginning a communication session with the Qbridge. The Qbridge will respond with an ACK packet |

| | | |
|---|---|---|
| ACK | 0x41 'A' | Acknowledges a packet. The first, and possibly only, byte of the data section is an ACK code. The ACK Codes are defined in ACK Code Table. Some commands may request data which is returned in the ACK packet. This data is documented with the individual commands and appears immediately following the ACK code.<br><br>The Packet ID field of an ACK packet works differently than for other packets. The Packet ID will the ID that the host transmitted |
| MID Filter Enable | 0x42 'B' | Enables or disables MID filtering. At startup, MID filtering is ON and all MIDs are set to filtered (i.e. no J1708 packets will be transmitted to the host until filters are disabled or MIDs are set to on)<br><br>The data field of this packet is one byte:<br>• 0x00: Disable MID filtering – all J1708 packets will be passed to the host<br>• 0x01: Enable MID filtering – only those J1708 packets whose MIDs are configured to pass through will be passed through to the host. |
| Set MID On/Off | 0x43 'C' | Adds or Removes a MID from the Qbridge filter list. Only applies if MID filtering is enabled.<br><br>The data portion of this packet consists of two sections. A boolean indicating whether MIDs are to be filtered, and a list of MIDs which will be affected.<br><br>The first data byte indicates whether this MID will be filtered or not.<br>• 0x00: MID off. If MID filters are enabled, messages with this MID will not be passed to the host<br>• 0x01: MID on. Messages with this MID will be passed to the host<br><br>The remaining data bytes are a list of MIDs. Each MID is 2 bytes (little endian). If you send the special value 0xFFFF in this list, then this will set the value for all MID filters. The packet length is used to determine the number of items in the list. Only values from 0-511 and 0xFFFF are allowed in this list. Any other values will be rejected, and cause the Qbridge to reply to this packet with an ACK_INVALID_DATA |

| | | |
|---|---|---|
| Send J1708 Packet | 0x44 'D' | Transmits a J1708 packet on the J1708 bus.<br><br>The data portion of this packet contains the following information:<br>• 1 Byte: The J1708 packet priority. This is an integer value from 0x01 to 0x08.<br>• <21 Bytes: The data portion of the J1708 packet, including the MID and PID value(s). The J1708 checksum will be automatically calculated by the Qbridge. This must be 20 or fewer bytes.<br><br>The Data field of an ACK for this packet will contain three pieces of information:<br>• Ack Code (1 byte, code) (documented under the Ack packet)<br>• Free Buffers (1 byte, integer): The number of free J1708 packet buffers on the Qbridge.<br>• J1708PacketID (4 byte, integer): The packet J1708 packet ID assigned to the packet. Only useful if J1708 Transmission confirmation is turned on. Little endian format<br><br>If the Qbridge was unable to add the packet, the J1708PacketID will be -1, and the Ack Code will be ACK_UNABLE_TO_PROCESS. |
| Receive J1708 Packet | 0x45 'E' | This packet will be initiated by the Qbridge and must be acknowledged by the host device. This indicates that the Qbridge has received a valid J1708 packet, and, if MID filters are enabled, that the packet has passed the MID filter.<br><br>The data field of this packet will contain the contents of the J1708 packet, less the checksum byte. (J1708 packets with an incorrect checksum are discarded by the Qbridge).<br><br>The host device should send an ACK with an ACK_OK as the data field in response to this message |
| Enable Transmit Confirm | 0x46 'F' | Enables or disables Transmission confirmation. When enabled, the Qbridge will send a Transmit Confirm command to the host whenever a packet has been successfully placed on the bus. This is a global setting that applies to ALL data busses.<br><br>This command contains 1 byte of data<br>• 0x00 Disable transmission confirmation. This is the default state<br>• 0x01 Enable transmission confirmation. |

| | | |
|---|---|---|
| Transmit Confirm | 0x47 ‘G’ | Initiated by the Qbridge to the host. If Transmission confirmation is enabled, then the Qbridge will send this command to the host each time a packet has been transmitted. Like all messages initiated by the Qbridge, this message must be acknowledged.<br><br>This data portion of this command contains 5 bytes.<br>• The first byte is a success/failure flag (0x00 =the packet was unable to be transmitted, 0x01=the packet was successfully placed on the bus)<br>• The remaining 4 bytes are an integer representing the packet identifier in little endian format. This is the value that was returned in the ACK response to the “Send J1708 Packet” or “Send CAN Packet” commands. |
| Upgrade Firmware | 0x48 ‘H’ | After ACKing this packet, the Qbridge will reset into the bootloader and await a firmware upgrade. |
| Reset QBridge | 0x49 ‘I’ | Causes the QBridge to reset. |
| Info Req | 0x2A ‘*’ | Transmit device info out the debug port of the Qbridge. Intended for internal QSI use only.<br><br>This contains 1 byte of information detailing which information you are interested in.<br>• 0: General Details<br>• 1: J1708 log information (debug builds only)<br>• 2: MID state information (debug builds only) |

| | | |
|---|---|---|
| Raw J1708 | 0x2B '+' | Raw J1708 Packet – Transmits a raw J1708 packet out the J1708 port. The user application is required to provide all checksum, MID and other data. Primarily intended for internal QSI use. The data portion of the packet is a J1708 packet (including MID and checksum). The maximum supported length is 21 bytes (including MID and checksum).<br><br>The Data field of an ACK for this packet will contain three pieces of information:<br>• Ack Code (1 byte, code) (documented under the Ack packet)<br>• Free Buffers (1 byte, integer): The number of free J1708 packet buffers on the Qbridge.<br>• J1708PacketID (4 byte, integer): The packet J1708 packet ID assigned to the packet. Only useful if J1708 Transmission confirmation is turned on.<br><br>If the Qbridge was unable to add the packet, the J1708PacketID will be -1, and the Ack Code will be ACK_UNABLE_TO_PROCESS. The Qbridge will select the J1708 priority for this packet<br><br>This packet is for internal QSI use only. |
| Request Raw Packets | 0x2C ',' | Request that the Qbridge transmit raw packets received from its data bus(es). When enabled, the Qbridge will forward all packets received regardless of whether they have a valid checksum or pass filtering. Also, the checksum byte(s) will not be removed, but will be included in the transmissions.<br><br>This request contains one byte of data :(boolean)<br>True: Enable raw mode<br>False: Disable raw mode (default) |
| J1708 Echo | 0x2D '-' | Request that the Qbridge echo transmitted packets back over the 232 serial port upon transmission.<br><br>This request contains one byte of data :(boolean)<br>True: Enable Echo<br>False: Disable Echo (default) |

| | | |
|---|---|---|
| Send CAN Packet | 0x4A‘J’ | Transmits a CAN packet on the CAN bus.<br><br>The Data field of this packet contains the following information:<br>• 1 Byte: The CAN Message Type. 0x00 for Standard CAN and 0x01 for Extended CAN.<br>• 2 or 4 Bytes: CAN Identifier, for Standard CAN the identifier is 2 bytes long, else it is 4 bytes long.<br>• <=8 Bytes: Message Data<br><br>The Data field of an ACK for this packet will contain three pieces of information:<br>• Ack Code (1 byte, code) (documented under the Ack packet)<br>• Free Buffers (1 byte, integer): The number of free CAN packet buffers on the Qbridge.<br>• PacketID (4 byte, integer): The packet CAN packet ID assigned to the packet. Only useful if CAN Transmission confirmation is turned on. Little endian format.<br><br>If the Qbridge was unable to add the packet, the PacketID will be -1, and the Ack Code will be ACK_UNABLE_TO_PROCESS. |
| Receive CAN Packet | 0x4B ‘K’ | This packet will be initiated by the Qbridge and must be acknowledged by the host device. This indicates that the Qbridge has received a valid CAN packet.<br><br>The data field of this packet will contain the contents of the CAN packet in the format:<br>• 1 Byte: The CAN Message Type. 0x00 for Standard CAN and 0x01 for Extended CAN.<br>• 2 or 4 Bytes: CAN Identifier, for Standard CAN the identifier is 2 bytes long, else it is 4 bytes long.<br>• <=8 Bytes: Message Data<br><br>The host device should send an ACK with an ACK_OK as the data field in response to this message |

| | | |
|---|---|---|
| CAN control packet | 0x4C ‘L’ | Provides control of certain aspects of the CAN controller within the Qbridge. The Data field of this packet contains the following information:<br>• 1 Byte: The CAN Control Message Type. ‘b’ to control baud rate, ‘d’ to control debug mode, ‘r’ to re-initialize after CAN bus error causes can controller to automatically enter the ‘busoff’ state, ‘n’ to notify when busoff state is entered, ‘f’ to set up a CAN filter, ‘g’ to retrieve the current filter list, ‘e’ to disable CAN filters.<br>• 0-Nbytes: Dependant upon first byte.<br>• for further details, see the CAN control packet description later in this document<br>• byte[0] = ‘b’ (baud rate setup)<br>• byte[0] = ‘d’ (debug setup)<br>• byte[0] = ‘r’ (recovery begin after CAN bus fault)<br>• byte[0] = ‘e’ (filter enable/disable)<br>• byte[0] = ‘f’ (filter setup)<br>• byte[0] = ‘g’ (filter list report)<br>• byte[0] = ‘n’ (bus off event notification)<br>• byte[0] = ‘a’ (auto recovery attempt from bus fault)<br>• byte[0] = ‘i’ (CAN info, same as option 0x02 of Get Information Command 0x4d ‘M’) –NOTE: This is not yet implemented |
| CAN Bus Error | 0x4E ‘N’ | This packet will be initiated by the Qbridge and must be acknowledged by the host device. This indicates that the Qbridge has detected a CAN bus error condition and cannot continue to process CAN send packet commands. In order to recover from the bus error condition external action may be required as this is most likely caused when the bus is shorted to ground or wires are crossed. Once the bus error no longer exists, it will be necessary to issue the CAN control packet with the ‘r’ option to restore the Qbridge CAN controller to a state where it can send and receive CAN packets.<br><br>There is currently no data field defined for this packet |
| Get Information | 0x4D ‘M’ | Return device information from the Qbridge.<br>This contains 1 byte of information detailing which information you are interested in.<br>• 0x00: General Details (version, build date)<br>• 0x01: J1708 information (see description below)<br>• 0x02: CAN information (see description below)<br>• 0x03: re-arm the overflow detection logic (see description below) |

| | | |
|---|---|---|
| Enable Advanced Receive Mode | 0x4F ‘O’ | Enables or disables alternative receive mode. When enabled, all CAN/J1708 receive packets will be forwarded to the host device without waiting for ACKs and received ACKs for these packets will be ignored.<br><br>The data field of this packet is one byte:<br>• 0x00: Disable advanced receive mode - waits for ACK from host device before sending next packet.<br>• 0x01: Enable advanced receive mode - all receive data passed through and ACKs ignored. |

**ACK Codes**

The following ACK codes are defined

| ACK Code | Value | Description |
|---|---|---|
| ACK_OK | 0x30 ‘0' | The packet was received and processed normally |
| ACK_DUPLICATE_PACKET | 0x31 ‘1' | A duplicate packet was received |
| ACK_INVALID_PACKET | 0x32 ‘2' | The packet was incorrectly formed, incomplete or had an invalid CRC. |
| ACK_INVALID_COMMAND | 0x33 ‘3' | An unknown or invalid command was received |
| ACK_INVALID_DATA | 0x34 ‘4' | The data field for the command was invalid |
| ACK_UNABLE_TO_PROCESS | 0x35 ‘5' | The device was unable to handle the request at this time. |
| ACK_BUS_FAULT | 0x36 ‘6' | The CAN packet was unable to be added to the CAN transmit queue due to the CAN controller being in a bus fault condition. The device will begin the auto recovery sequence after reporting this error if autorecovery is enabled otherwise it will be necessary to send a bus recovery command (‘Lr’) to the QBridge. |
| ACK_OVERFLOW_OCCURRED | 0x37 ‘7' | Reported for any command after a queue overflow has occurred in either the CAN receive queue or the J1708 receive queue. This will be reported once for either queue and will not be reported again unless reset (see the get information 0x4D ‘M” command). The data field sent with this packet will indicate which bus had the overflow event (0x01 = CAN, 0x02 = J1708, 0x03 = both) |

**CAN Control Packet (command code 'L' - 0x4C)**

Provides control of certain aspects of the CAN controller within the Qbridge. The Data field of this packet contains the following information:

- 1 Byte: The CAN Control Message Type. 'b' to control baud rate, 'd' to control debug mode, 'r' to re-initialize after CAN bus error causes can controller to automatically enter the 'busoff' state, 'n' to notify when busoff state is entered, 'f' to set up a CAN filter, 'g' to retrieve the current filter list, 'e' to disable CAN filters.
- 0-Nbytes: Dependant upon first byte.
- byte[0] = 'b' (baud rate setup)
  - byte[1] = '0' use the default baud rate (J1939 specifies 250K)
  - byte[1] = '1' set the CAN baud rate to 1Meg
  - byte[1] = '2' set the CAN baud rate to 500K
  - byte[1] = '3' set the CAN baud rate to 250K
  - byte[1] = '4' set the CAN baud rate to 125K
- byte[0] = 'd' (debug setup)
  - byte[1] = 'L' set the CAN controller to Loop Back mode
  - byte[1] = 'S' set the CAN controller to Silent mode
  - byte[1] = 'H' set the CAN controller to Hot Self Test mode
  - byte[1] = 'N' set the CAN controller to Normal mode
- byte[0] = 'r' (recovery begin after CAN bus fault)
  - byte[1] = is a bit mapped value indicating which recovery actions to take
    - bit 0 = restart the CAN controller hardware (sets the run bit)
    - bit 1 = reset all defaults (Disables CAN transmit confirm, Disables CAN bus fault notification, Enables firmware auto recovery from CAN bus faults, Disables CAN message filters, Clears all queues (as if bits 3, 4, and 5 were set))
    - bit 2 = reinitialize the CAN controller hardware (same as hardware reset)
    - bit 3 = Clear the CAN Transmission Queue (all pending transmissions that did not occur before the bus fault will be ignored)
    - bit 4 = Stop any CAN Transmissions that are actively being transmitted (note, certain bus conditions such as when there are no other CAN devices will be continuously retransmitted (until ACK'd by another device)... this option allows a way to cancel such a retransmission)
    - bit 5 =Clear the CAN Receive Queue (all pending packets that have not yet been sent to the host will be ignored)
- byte[0] = 'e' (filter enable/disable)
  - byte[1] = 0 disable all CAN filters (all can messages will be transmitted back to host)
  - byte[1] = 1 enable the CAN filters (only those messages that match the criteria will be transmitted back to the host, if no filters have been setup, nothing will be transmitted back to the host)
- byte[0] = 'f' (filter setup)
  - byte[1] = 0 disable the following list of filters
  - byte[1] = 1 enable the following list of filters
    - bytes[2-n] =list of identifiers and masks to use for filtering incoming CAN messages. Multiple filters may be sent in one command. The list may contain both standard and extended identifiers. The list is comprised of 1 byte indicating standard(=0) or extended CAN identifier, then either 2 bytes (for standard) or 4 bytes (for extended)

of identifier information (in little endian format), then either 2 bytes (for standard) or 4 bytes (for extended) of mask information. The filter operation that is performed is as follows: if( received_CAN_identifier & filter_mask == filter_identifier) then send information received to host. The maximum number of filters that can be enabled is 25.

- byte[0] = ‘g’ (filter list report) no other bytes are required. This option returns a list (as described in the filter setup) of the enabled filters. It should be noted that if filters have been globally disabled (such that ALL CAN messages are transmitted to host), this option will still return the list of filters that were present when filters were disabled.
- byte[0] = ‘n’ (bus off even notification)
  - byte[1] = 0 disable CAN bus off event notification
  - byte[1] = 1 enable CAN bus off event notification
- byte[0] = ‘a’ (auto recovery attempt from bus fault)
  - byte[1] = 0 disable CAN bus off automatic recovery
  - byte[1] = 1 enable CAN bus off automatic recovery
- byte[0] = ‘i’ (CAN info, same as option 0x02 of Get Information Command 0x4d ‘M’)

Notes:
Loop Back Mode turns the Qbridge’s transmitter back to the receiver. Data transfer from the Qbridge can be observed on the CAN bus. Any actual data on the CAN bus is ignored by the Qbridge, only data transmitted from the Qbridge can be seen as received data by the Qbridge
Silent Mode disables the Qbridge’s transmit signal. The Qbridge is still able to receive messages from the CAN bus. This mode may be used to analyze a CAN bus without disturbing the bus
Hot Self Test Mode is the combination of silent and loop back, thus the Qbridge can perform self test functions while connected to a CAN bus system without disturbing other devices on the bus. However, during this mode, the Qbridge will not receive messages from any of the other devices.
Normal Mode turns on the transmitter and receiver. Any received messages must pass the normal filtering before being passed on via Receive CAN packet.

Example filter setup command:

if we want to see standard CAN messages with the two least significant bits set to 10b and we want to see extended CAN messages with the three most significant bits set to 101b we would send the following:

```
'L', 'f', 1, 0, 02, 00, 03, 00, 01, 00, 00, 00, 14, 00, 00, 00, 1c
 |    |   |  |   |   |   |   |   |   |   |   |   |   +---+---+---+---- extended mask =0x1c000000
 |    |   |  |   |   |   |   |   |   +---+---+---+---- extended identifier=0x14000000
 |    |   |  |   |   |   |   |   +---- type specifier... following information is 'extended'
 |    |   |  |   |   |   +---+---- standard mask = 0x0003
 |    |   |  |   +----+---- standard identifier = 0x0002
 |    |   |  +---- type specifier... following information is 'standard'
 |    |   +---- enable the filters in this list
 |    +---- sub command is the 'filter setup' option
 +---- command is 'CAN control' command
```

## Get Information Packet (command code 'M' - 0x4D)

**Get Version Information Packet (command code 'M' - 0x4D— option 0)**

- byte[0] = 0x00: General Details - this option returns a string indicating firmware revision and build date/time and also the bootloader version and build date/time.

**Get J1708 Information Packet (command code 'M' - 0x4D— option 1)**

- byte[0] = 0x01: J1708 information - this option returns 14 words (56 bytes) of information (in little endian format) pertaining to the J1708 firmware. The words are in the following order:
  - word 0 = J1708 Received packet count - counts the number of valid (good checksum, but independent of filtering) J1708 packets received by this device
  - word 1 = J1708 Wait for Busy Bus Count - increments each time this device attempts to send a message but must wait because the bus has not been idle long enough. This will get incremented multiple times for the same message as the firmware polls for the conditions to be correct for sending
  - word 2 = J1708 Collision Count - incremented each time this device detects a transmission collision between itself and another transmitting device.
  - word 3 = J1708 Dropped Rx Count - incremented when a message is received but the queue of messages to send to the host is full. In general, since the host interface is faster than the J1708 interface, this should never happen. However, if the host is slow in responding to the transmitted messages, it can happen.
  - word 4 = J1708 Dropped Tx Confirm Count - incremented when a message has been transmitted that should generate a transmit confirm but the queue of message to send to the host is full. Since the transmit confirms utilize the same queue as the received messages this count will have the same conditions as the Dropped Rx Count
  - word 5 = J1708 Dropped packets from host (due to TX queue full) - incremented each time a request is received from the host to transmit a J1708 packet but the Tx Queue is full. The command to transmit will receive an ACK_UNABLE_TO_PROCESS and this counter will be incremented
  - word 6 = J1708 MID Filter enabled (boolean indicating filter enable/disable state)
  - word 7 = J1708 Transmit Confirm (boolean indicating J1708 transmit confirm enable/disable state)
  - word 8 = J1708 RxQueue Overflowed
  - word 9 = J1708 Bus transitions detected since last time this command was issued (this may be used to detect a stuck bus)
  - word 10 = J1708 Current bus state (non zero if the bus is asserted, zero if idle)
  - word 11 = J1708 Number of free Tx buffers (queue positions)
  - word 12 = J1708 Number of free Rx buffers (queue positions)
  - word 13 = J1708 overflow reported (indication of state that keeps track of whether or not an overflow condition has been detected and reported)
  -

**Get CAN Information Packet (command code 'M' - 0x4D— option 2)**

- byte[0] = 0x02: CAN information - this option returns 30 words (120 bytes) of information (in little endian format) pertaining to the CAN firmware. The words are in the following order:
  - word 0 = Tx Count - number of CAN packets transmitted by this device
  - word 1 = Rx Count - number of CAN packets seen on the bus by this device
  - word 2 = Stuff Error Count - number of times this device has seen a CAN message with

more than 5 equal bits in a sequence
- word 3 = Form Error Count - number of times this device has received a message where a fixed format part of the received frame have the wrong format
- word 4 = Ack Error Count - number of times the message transmitted by this device was not acknowledged by another device.
- word 5 = Bit 1 Error Count - number of times this devices attempted to send a receissive bit but monitored a dominant bit (i.e.a collision occured on the bus)
- word 6 = Bit 0 Error Count - number of times this device attempted to send a dominant bit but monitored a recessive bit. During bus off recovery, this will increment each time 11 recessive bits has been monitored.
- word 7 = CRC Error Count
- word 8 = Lost Message Count
- word 9 = CAN Rx interrupt queue overflowed - boolean indicating that data was received (passed filtering) but the queue of messages to send to the host was full.
- word 10 =CAN Rx interrupt queue overflowed count - incremented each time the above condition is detected (either for a received message or a successful transmission that requires confirmation back to the host).
- word 11 =CAN transmit confirm - boolean indicating transmit confirmation is enabled
- word 12 =CAN Bus Off Notify - boolean indicating bus off notification is enabled
- word 13 = CAN Auto Restart - boolean indication that the firmware should automatically attempt to recover from a bus off condition
- word 14 = CAN Auto Recovery Attempt Count
- word 15 = CAN Rx Wait for host count
- word 16 = CAN Rx Bad Value Count - indicates an internal firmware problem
- word 17 = bus off interrupt count - number of times the bus off interrupt has occurred
- word 18 = bus off notify count - number of times the bus off notification has been generated (only incremented if bus off notification is enabled)
- word 19 = bus off want to notify host count - number of times the bus off notification would have been generated if notification was enabled
- word 20 = CAN bus error count
- word 21 = CAN dropped from host count
- word 22 = CAN Bus transition detected since last time this command was exectued
- word 23 = CAN Current Bus State - current state of the CAN bus (
- word 24 = CAN Free Tx Buffers
- word 25 = CAN Free Rx Buffers
- word 26 = CAN Baud Rate
- word 27 = CAN test mode
- word 28 = CAN Hardware error counters
- word 29 = CAN overflow reported

**Re-Arm Overflow Information Packet (command code 'M' - 0x4D— option 3)**

- byte[0] = 0x03: re-arm the overflow detection logic - reset J1708 overflow reported, J1708 RxQueue Overflowed, CAN overflow reported and CAN Rx interrupt queue overflowed variables.

The device maintains a queue of information received from each of the two busses (J1708 and CAN) in order to send that information to the host. Should the host fail to respond in a timely manner or if the CAN bus is very busy (since it's baud is more than 2 times that of the host bus) it is possible to receive more information than can be queued by this device. When that occurs, the host is notified upon receipt of the next command. No matter what the command (except an Ack to a previous 'Receive

1708 or Receive CAN') the device will terminate the command with ACK_OVERFLOW_OCCURRED. But because more data may be received (thus causing another 'overflow') before the host has time to do anything about the overflow condition, the device maintains a state machine to ensure that this status is only reported one time (independant of the number of messages received while there is no room in the queue). The state machine is observable thru the J1708 overflow reported and the CAN overflow reported variables. When set to 0 the state machine is armed and the first occurrence of an overflow will get reported the next time the host attempts to send a command. When set to 2, the state machine has reported the status and will not report another until reset by the Re-Arm Overflow Packet (command code 'M' - 0x4d – option 3).

## Qbridge Bootloader communications:

This describes the 232 serial protocol used by the Qbridge bootloader. The Qbridge bootloader is active when the main firmware has not been programmed, has become corrupted, or when a request to reprogram the main firmware has been accepted.

When the bootloader begins, the following string will be sent:

```
Entering QSI Qbridge Bootloader VERSION \r\n
Built  __DATE__  at  __TIME__ \r\n
```

Where VERSION is replaced by the actual version number (as of this writing, V0.006) and __DATE__ and __TIME__ are replaced by the actual date and time the bootloader firmware was built (as of this writing, Feb 02 2007 and 10:44:34

If the bootloader is invoked in response to a request to program the main firmware, the following string will preceed that described above:

```
NOTICE--received request from kernel to enter bootloader\r\n
```

Once these messages have been sent, the bootloader waits for communications from the host. The only communications expected from the host is the transmission of an S-record (.srec) file. Thus, the bootloader is looking for ascii strings terminated by a carriage return line feed sequence. A line may end in either CR (0x0d) followed by LF (0x0a) or simply with an LF (0x0a). Each line is evaluated individually as received and a good status (ACK) or error status (NAK) is returned. The ACK status is a single byte of value 0x06 sent by the Qbridge bootloader to the host. The NAK status is a single byte of value 0x15 sent by the Qbridge bootloader to the host. This is summarized in the following table.

| Qbridge Bootloader expects | Qbridge Bootloader response | |
|---|---|---|
| ascii strings (valid srec lines) seperated by CR/LF or just LF (ie 0x0d 0x0a or just 0x0a) | ACK | 0x06 |
| | NAK | 0x15 |

A NAK may be returned for the following reasons:

1. An invalid S-record is received
    a) the line does not begin with an S or s
    b) the length of the received line (minues the CR/LF) is not a multiple of 2
    c) the length as specified in the S-record does not match the length of the received line
    d) the S-record checksum is invalid
2. An unhandled S-record type is received
    a) the S-record type is something other than 0, 3, or 7
    b) the S-record module name (as specified record type 0) is NOT qbridge.srec

No status will be returned (i.e. the input will simply be ignored without any indication and without any action) if more than 256 characters have been received without finding the CRLF sequence or if extra characters are sent immediately following the CRLF sequence (or for example if the CRLF sequence is sent reversed... LFCR).

It should be noted that there is no timeout on the time between reception of characters. Received characters are held indefinitely until a CRLF sequence is seen (or more than 256 characters have been received or until power is cycled).

Using the above knowledge, it is possible to determine if the bootloader is active by sending an invalid SREC line and expecting a NAK in return. This is usually done in response to a failed (timeout) main firmware communication to determine if the the Qbridge is in bootloader mode.

If the bootloader is active (ie the Qbridge is in bootloader mode), sending the following sequence will cause the qbridge to return a NAK (0x15) status.

A\r\n

During transmission of the SREC file, each line will either receive an ACK or a NAK from the bootloader. The host device sending the SREC file shall wait for the bootloader to respond after each line of the SREC file is sent before sending the next line. A reasonable timeout may be implemented and the offending line may be retransmitted if desired. It should be noted that the module specifier (S-record type 0) takes significantly longer for the bootloader to respond than all others s-record types as it causes the entire main firmware section of flash (128K bytes) to be erased.

Note that in order to “patch” the main firmware, the SREC file should NOT include a module specifier as that causes the entire main firmware area of flash to be erased. Remember when patching... erased flash contains all ones (0xff) and programing is simply writing zeros.... ie you can change ones to zeros but you can’t change zeros to ones.